Computational Journalism: from Answering Questions to Questioning Answers and Raising Good Questions

by

You Wu

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Jun Yang, Co-Supervisor

_____
Pankaj K. Agarwal, Co-Supervisor

_____
Ashwin Machanavajjhala

_____
Cong Yu

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

Abstract

Computational Journalism: from Answering Questions to Questioning Answers and Raising Good Questions

by

You Wu

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Jun Yang, Co-Supervisor

_____
Pankaj K. Agarwal, Co-Supervisor

_____
Ashwin Machanavajjhala

_____
Cong Yu

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2015

Copyright © 2015 by You Wu
All rights reserved except the rights granted by the
Creative Commons Attribution-Noncommercial Licence

Abstract

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, e.g., is a claim "cherry-picking"? This thesis proposes a Query Response Surface (QRS) based framework that models claims based on structured data as parameterized queries. A claim is mapped to a point on the QRS. A key insight is that we can learn much about a claim by analyzing its neighborhood on QRS. This framework lets us formulate and tackle practical fact-checking tasks — reverse-engineering vague claims, and countering questionable claims — as computational problems.

Within the QRS-based framework, we take one step further, and propose a model as well as an efficient algorithm for finding high-quality claims of a given form from data, i.e., raising good questions, in the first place. This is achieved by using a limited number of high-valued claims to represent high-valued regions of the QRS.

Besides the general purpose high-quality claim finding problem, lead-finding can be tailored towards specific claim quality measures, also defined within the QRS framework. An example of uniqueness-based lead finding is presented for "one-of-the-few" claims, landing in interpretable high-quality claims, and an adjustable mechanism for ranking objects, e.g. NBA players, based on what claims can be made for them.

Finally, we study the use of visualization as a powerful tool of conveying results

of a large number of claims. An efficient two-stage sampling algorithm is proposed for generating input of 2d scatter plot with heatmap, evalutaing a limited amount of data, while preserving the two essential visual features, namely outliers and clusters.

A system prototype (iCheck) has been developed performing efficiently the rich class of fact-checking and lead-finding tasks studied in this thesis. We hope this could serve as a first step towards an end-to-end system empowering journalists and the public in combatting the "lies, d—ed lies, and statistics" that permeate our public life today.

Contents

List of Tables

List of Figures

Acknowledgements

I want to thank my co-advisors Jun Yang and Pankaj K. Agarwal for their guidance, support, and encouragement towards the completion of this dissertation. Their advice has always been insightful and prompt. I wholeheartedly appreciate their patience in showing me the right approach towards conducting rigorous and impactful computer science research.

I also thank Ashwin Machanavajjhala and Cong Yu for their service on my committee, and for their valuable comments and suggestions on this dissertation.

I also want to thank my collaborators on the work of this dissertation, and on the Computational Journalism project. Besides my co-advisors Jun and Pankaj, and one of my committe member Cong, they are Bill Adair, Peggy Li, Andrew Shim, Stavros Sintos, Emre Sonmez, Seokhyn Song, Brett Walenz, Eric Wu, Kevin Wu from Duke University; Naeemul Hassan, Chengkai Li, Afroza Sultana, Gensheng Zhang from the University of Texas at Arlington; Boulos Harb from Google Inc.

I thank the Shivnath Babu, Kamesh Munagala, and Xiaobai Sun of Duke Computer Science for their feedback on my research, and advice on conducting research in general. I also thank Marilyn Butler and the rest of the Duke Computer Science administrative and laboratory staff for helping me with all kinds of logistic details, and allowing me to better focus on my research.

I thank Cong Yu and Boulos Harb for hosting my first Google internship in 2013, and teaching me good engieering practice, which I found later to be particularly help-

ful in research and in developing the iCheck system. I also want to thank Min Wang and Haixun Wang for hosting my second Google internship in 2014, and teaching me how to conduct impactful research in an engineering research environment.

I want to thank my fellow graduate students and friends at Duke, for their constant support and inspiration. To name a few: Muhammad Ahsan, Qiang Cao, Balakrishnan Chandrasekaran, Liang Dong, Yuzhang Han, Botong Huang, Yezhou Huang, Zhiqiu Kong, Janardhan Kulkarni, Mayuresh Kunjir, Yuqian Li, Jiangwei Pan, Tianqi Song, Rishi Thonangi, Xixi Wang, Xiao Xiao, Bing Xie, Ke Xu, Xiaoming Xu, Albert Yu, Wuzhou Zhang, Xuting Zhao, and Yunjia Zhou.

I want to thank Ke Yi from the Hong Kong University of Science and Technology. He has been a mentor and a good friend. As a former student of Pankaj, he was a big part of why I joined Duke. I thank him for giving me the opportunity to meet and interact with all these amazing people in the past five years.

I acknowledge the funding that I received directly and through my co-advisors research grants. I thank the Duke Graduate School, the National Science Foundation, the Army Research Office, the U.S. Army Engineer Research and Developement center, the U.S.-Israel Binational Science Foundation, and HP Labs.

Finally, I want to thank my parents Yiming Wu ang Guangwei Shi for their constant support, confidence, and love. None of this would have been possible without their faith in me.

1

Introduction

Database research in the past has been devoted to the art of *answering* queries. There is much work on algorithms and techniques to speed up query evaluation, such as indexing data structures, pipelined execution, parallel evaluation, to name a few.

A query can be translated or presented in natural language as a claim of fact based on the underlying data. The art of *discerning* the "quality" of such claims and *asking* queries that lead to high-quality claims are critical to our understanding of data. But first, what does "quality" mean? Consider the following.

Example 1 (**Giuliani's Adoption Claim** (from `factcheck.org`))**.** During a Republican presidential candidates' debate in 2007, Rudy Giuliani claimed that "adoptions went up 65 to 70 percent" in the New York City "when he was the mayor." More precisely, the comparison 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 six-year periods? As it turns out (as shown in Figure 1.1), 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, i.e., 1994–1997 and 1998–2001, we will see that the total number of adoptions in fact decreased by 1%.[1]

Example 2 (**Vote Correlation Claim** (from `factcheck.org`))**.** A TV ad in the 2010 elections claimed that Jim Marshall, a Democratic incumbent from Georgia "voted the same as Republican leaders 65 percent of the time."[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 percent, which is not very high considering the fact that even the Democratic Whip, Jim Clyburn, voted 44 percent of the time with Boehner during that period. Basically, many votes in Congress are not as controversial as the public would think!

For both claims above, we can verify their correctness using SQL queries over reliable, structured datasets available to the public. Database systems are good at verifying whether these claims are correct, but from the above discussion, it is obvious that assessing claim quality involves much more than testing correctness. Indeed, both claims above 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 claims above are in-

[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 US 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.

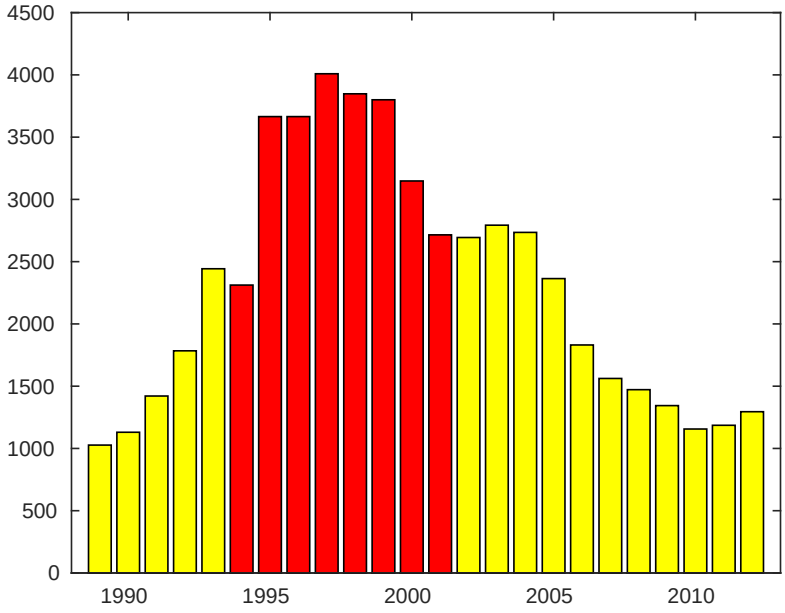


FIGURE 1.1: New York City adoptions by year, 1989–2012. Giuliani's years in red.

tuitive, deriving them requires considerable skill and effort. Not all users think as critically as we would hope. Not all users who are suspicious of a claim have the time or expertise to conduct further data analysis. How do we make this process easier and more effective?

For example, is there any hope of formalizing the intuitions behind Example 1, and automating the assessment of claim quality? To help illustrate claim quality to lay users, we often need "counterarguments"; e.g., for Giuliani's claim we highlighted that *"if we compare Giuliani's first and second 4-year terms, i.e., 1994–1997 and 1998–2001, we will see that the total number of adoptions in fact decreased by 1%."* Can we automatically generate such counterarguments? Many claims, such as Giuliani's and Marshal's, are vaguely stated (often intentionally). Before checking a vague claim, can we automatically "reverse-engineer" it to recover missing details? Given measures of claim quality, can we automatically find high-quality claims from data? Can such measures help us identify interesting objects (e.g. NBA players) as

leads to further investigation? Can such lead-finding be carried out without specified quality measure? Can visualization help explore the space of claims more efficiently?

These problems has many applications in domains where assessments and decisions are increasingly driven by data. *Computational journalism* (Cohen et al., 2011a,b) is one domain with the most pressing need for better fact-checking techniques. With the movement towards accountability and transparency, the amount of data available to the public is ever increasing. Such data open up endless possibilities for empowering journalism's watchdog function—to hold governments, corporations, and powerful individuals accountable to society. However, we are facing a widening divide between the growing amount of data on one hand, and the shrinking cadre of investigative journalists on the other. Computing is a key to bridge this divide.

Contributions. This dissertation serves as a first step towards computational solutions, and eventually a suite of computational tools, for solving a wide range of important data related journalistic tasks, which will allow journalists and the general public to check, find, and explore claims based on structured data at increased efficiency and effectiveness.

The key insight is that much can be learned about a claim by "perturbing" it in interesting ways. Thus, we model a claim as a parameterized query over data, whose result would vary as we change its parameter setting, forming a (high-dimensional) surface which we call the ***q****uery* ***r****esponse* ***s****urface (QRS)*. With model components such as *parameter sensibility* and relative *result strength*, we show how to how the series of questions above as computational tasks within the model.

We first present the QRS based framework, show how to define some claim quality measures, and model the tasks of *finding counterarguments* and *reverse-engineering vague claims* as computational problems within the framework (Chapter 2). For the two fact-checking tasks, we propose meta algorithms that separates domain knowl-

edge from computaional techniques.

Next, we show examples of instantiating the modeling framework and the meta algorithms using claims generalized from Examples 1 and 2 (Chapter 3). Extensive experiements are performed to validate the ability of the framework in modeling fact-checking tasks and producing meaningful result, as well as the efficiency of the computational techniques. Chapters 2 and 3 are based on joint work with Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu (Wu et al., 2014b).

We show how to find interesting leads by choosing a small number of representative points to represent high-response regions of the response surface (or its variantions) (Chapter 4). Our method differs from existing approaches by considering jointly several aspects of high-quality representative leads, such as the quality of a representative itself, the quality of its neighborhood, and spatial diversity of representatives.

We study the lead-finding problem driven by one specific quality measure, namely *uniqueness*, on one common types of claims in sports analytics and other fields, called "one-of-the-few" (Chapter 5). Beyond effciently generating claims with interpretable quality in uniqueness, we propose an adjustable mechanism for ranking objects, e.g. NBA players, based on the set of "one-of-the-few" claims that can be made for them. This is based on joint work with Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu (Wu et al., 2012).

Finally, we consider the possibility of utilizing visualiztion as a more powerful tool than text to explore the space of claims (Chapter 6). In order to support responsive exploration of such type, an efficient two stage sampling algorithm is proposed for generating input of 2d scatter plot with heatmap, evalutaing a limited amount of data, while preserving the two essential visual features, namely outliers and clusters. This is based on joint work with Boulos Harb, Jun Yang, and Cong Yu (Wu et al., 2015).

2

Fact-Checking: Framework, Problems, and Algorithms

2.1 Introduction

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

Besides numerical measures of claim quality, *counterarguments* are critical in helping users, especially a non-expert public audience, understand why a claim has poor quality. As examples, we counter Giuliani's adoption claim with the argument that "*... 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 and 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 above challenges, we propose a fact-checking framework general enough to handle various types of claims and fact-checking tasks. The key insight is that a lot of fact-checking can be accomplished by "perturbing" the claims in interesting ways. Thus, we model a claim as a parameterized query over data, whose result would vary as we change its parameter setting, forming a (high-dimensional) surface which we call the ***q****uery* ***r****esponse* ***s****urface (QRS)*. In conjunction with QRS, we introduce the notions of relative *result strength*, which captures how a perturbation strengthens or weakens a claim, and of *parameter sensibility*, which captures how natural and relevant a perturbation is in the context of the claim being checked. To illustrate the modeling power of this framework:

- We show that many intuitive measures of claim qualities can be defined naturally. For example, a claim has low "robustness" if sensible perturbations of claim parameters lead to substantially weaker, or even opposite, conclusions.
- We show how to formulate fact-checking tasks—such as *finding 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) and *time series similarity claims* (generalization of the Marshall-Boehner vote correlation claim in Example 2).

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

- We propose "meta" algorithms 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 that can be plugged into the meta algorithms above to enable fact-checking in real time for interactive users.

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

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 3.5.

Table 2.1: 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

2.2 Modeling Framework for Fact-Checking

2.2.1 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.

Parameterized Query Templates, QRS, and Claims Let $q : \mathcal{P} \mapsto \mathcal{R}$ 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.[1] The ***q**uery **r**esponse **s**urface* of q, or *QRS* for short, is the "surface" $\{(p, q(p)) \mid p \in \mathcal{P}\}$ in $\mathcal{P} \times \mathcal{R}$.[2]

[1] 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 3.5 briefly discusses this possibility.

[2] 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

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 2.1, 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.

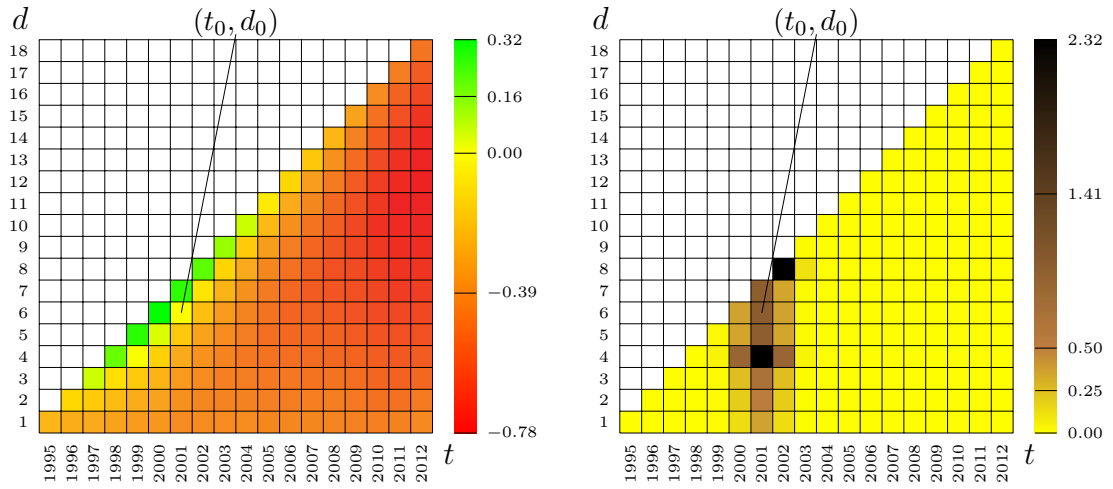
For example, to check Giuliani's claim in Example 1, suppose we have a table `adopt(year, number)` of yearly adoption numbers. The parameterized query template here can be written in SQL, with parameters w (length of the period being compared), t (end of the second period), and d (distance between the 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, resp.) than r_0. We require that $\mathsf{SR}(r; r) = 0$. For example, we let $\mathsf{SR}(r; r_0) = r/r_0 - 1$ for Giuliani's claim.

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

FIGURE 2.1: Perturbing t (end of the second period) and d (distance between periods) in Giuliani's claim while fixing $w = 6$ (length of periods). Note the constraint $t - d - w \geqslant 1988$; 1989 is when the data became available.

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 2.1a 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

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 2.1b illustrates the relative sensibility of parameter settings for checking Giuliani's claim (again, we fix w and vary only t and d). Darker shades indicate higher sensibility. The interaction of naturalness and relevancy results in generally decaying sensibility scores around $(t_0, d_0) = (2001, 6)$ (because of relevancy), but with bumps when $d = 4$ and $d = 8$ (because of naturalness—the New York City mayor has 4-year terms). Intuitively, portions of the QRS over the high-sensibility regions of the parameter space are more "important" in checking the claim. See Section 3.2 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 3.3 for more details.

2.2.2 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 $\mathsf{SR}(q(p); r_0) < 0$; i.e., it weakens the original claim. For example, Figure 2.1a shows counterarguments to Giuliani's claim in orange and red; they result in a lower percentage of increase (or even decrease) than what Giuliani claimed. Since there may be many counterarguments, we are most interested in those weakening the original claim significantly, and those obtained by highly sensible parameter perturbations. There is a trade-off between parameter sensibility and result strength: if we consider counterarguments with less sensible parameter perturbations, we might be able to find those that weaken the original claim more. Finding counterarguments thus involves bicriteria optimization. We define the following problems:

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

(CA-$\tau_{\mathfrak{P}}$) 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. This problem is formulated below:

(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) $\mathsf{SP}(p; p_0) \geqslant \mathsf{SP}(p'; p_0)$ (i.e., p is more sensible than or equally sensible as p'); ii) $\mathsf{SR}(q(p); r_0) \leqslant \mathsf{SR}(q(p'); r_0)$ (i.e., p weakens the original claim as much as or more than p'); and iii) inequality is strict for at least one of the above. A *Pareto-optimal* counterargument is one that is dominated by no counterarguments.

Reverse-Engineering Vague Claims As motivated in Section 2.1, many claims are not stated precisely or completely; e.g., Giuliani's adoption claim omits its parameter values and rounds its result value. We still represent a vague claim by $\langle q, p_0, r_0 \rangle$. However, p_0 and r_0 are interpreted differently from other problem settings. Here, r_0 may be an approximation of the actual result. For parameter values mentioned explicitly by the claim, p_0 sets them accordingly. On the other hand, for omitted parameters, p_0 sets them to "reasonable" values capturing the claim context. For example, Giuliani's adoption claim does not state the periods of comparison. We simply use 1993–1993 and 2001–2001 for p_0, i.e., $(w_0, t_0, d_0) = (1, 2001, 8)$, to capture the claim context that Giuliani was in office 1994–2001 (p_0 represents the comparison between two 1-year window, 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 p *dominates* another one p' if i) $p \succeq^{p_0} p$; ii) $|\mathsf{SR}(q(p); r_0)| \leqslant |\mathsf{SR}(q(p'); r_0)|$; and iii) inequality is strict for at least one of the above. A *Pareto-optimal* 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),[3] 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.

[3] 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.

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.

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

$$\sum_{p \in \mathcal{P}} \mathsf{SP}(p; p_0) \cdot \mathsf{SR}(q(p); r_0). \tag{2.1}$$

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

$$\exp\left(-\sum_{p \in \mathcal{P}} \mathsf{SP}(p; p_0) \cdot (\min\{0, \mathsf{SR}(q(p); r_0)\})^2 \right). \tag{2.2}$$

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

$$\frac{1}{|\mathcal{P}|} \cdot \sum_{p \in \mathcal{P}} \mathbf{1}\big(\mathsf{SR}(q(p); r_0) < 0\big). \tag{2.3}$$

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—e.g., the Marshall-Boehner vote correlation claim in Example 2, or *one-of-the-few* claims in (Wu et al., 2012) and Chapter 5—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—e.g., 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 ariability 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 1.1) 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.

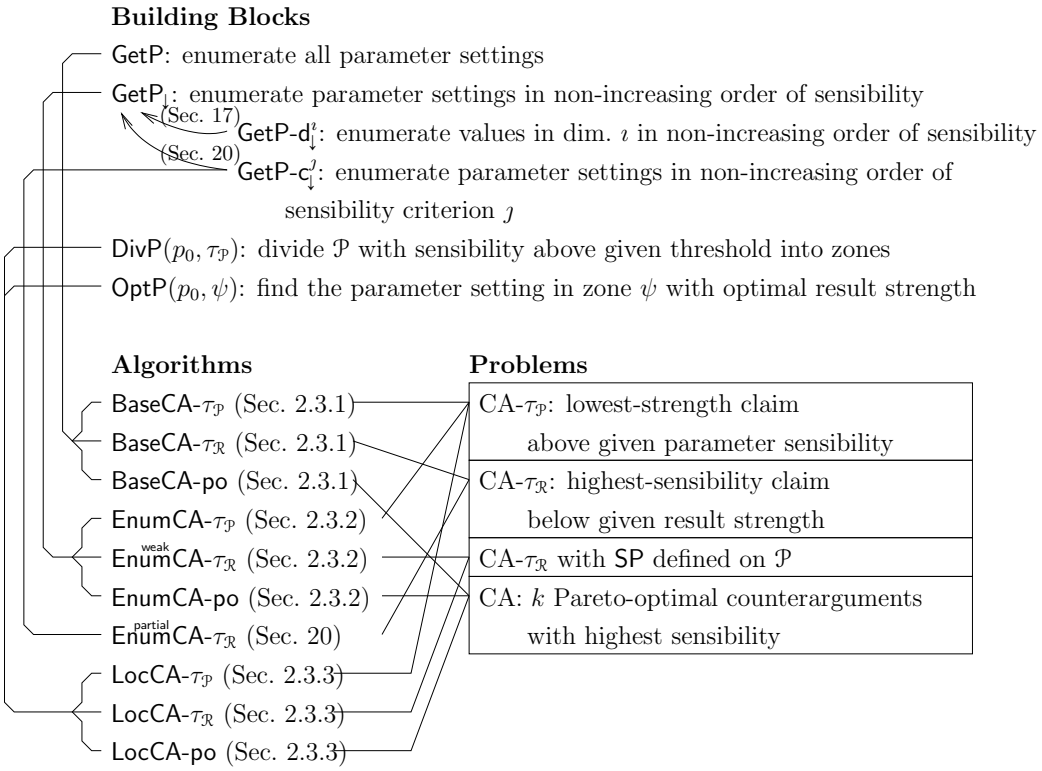


FIGURE 2.2: Notations for the algorithmic framework.

2.3 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 ones specialized for particular types of claims. However, more efficient algorithms are only possible by exploiting specific properties of the claims. To gain efficiency without sacrificing generality, we develop a series of "meta" algorithms (also summarized in Figure 2.2):

- Our baseline algorithms (Section 2.3.1) 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 al-

gorithms assume more powerful building blocks: functions that generate parameter settings in decreasing sensibility order (Section 2.3.2), and those that support a locus approach for searching the parameter space (Section 2.3.3). Instantiations of such building blocks for claims generalizing Examples 1 and 2 will be presented in Sections 3.2.2 and 3.3.2, 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 3.2.2 and 3.3.2 presents examples of how plugged-in preprocessing helps with checking claims in Examples 1 and 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 upfront, 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.

2.3.1 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).

BaseCA-$\tau_{\mathcal{P}}$ 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 $t \leqslant |\mathcal{P}|$ 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 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 2-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 $t \leqslant |\mathcal{P}|$ 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.

BaseCA-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.

2.3.2 Ordered Enumeration of Parameters

We now develop algorithms that take advantage of functions that enumerate, on demand, all possible parameter settings in non-increasing 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 2.3.2) when the parameter sensibility function $\mathsf{SP}(p; p_0)$ is defined, and an improved version of GetP, which we denote by $\mathit{GetP}_{\downarrow}$, is available for generating parameter settings with high SP first.

The second case (Section 17) extends the first, and is rather common for multi-dimensional parameter spaces. Here, instead of requiring $\mathsf{GetP}_{\downarrow}$, we require, for each dimension $\imath$, a function $\mathit{GetP\text{-}d}^{\imath}_{\downarrow}$ for enumerating the values in this dimension in non-increasing order. We assume this order is consistent with the overall sensibility: i.e., given a parameter setting p, replacing its value for dimension $\imath$ with one that appears earlier in this order cannot decrease $\mathsf{SP}(p; p_0)$. We give a general procedure that uses the single-dimensional $\mathsf{GetP\text{-}d}^{\imath}_{\downarrow}$'s to implement a single multidimensional $\mathsf{GetP}_{\downarrow}$, so we can then apply the algorithms for the first case above. Giuliani's adoption claim can be handled with this approach, as we will see in Section 3.2—we need to specify how to enumerate values in each of the three dimensions 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 20), again, no single $\mathsf{GetP}_{\downarrow}$ is given, but parameter settings can be compared according to multiple criteria. For each criterion $\jmath$, a function $\mathit{GetP\text{-}c}^{\jmath}_{\downarrow}$ exists to enumerate parameter settings in order according to $\jmath$. The $\mathsf{GetP\text{-}c}^{\jmath}_{\downarrow}$'s together define a partial order $\preceq^{p_0}$ on $\mathcal{P}$. For example, as we will see in Section 3.3, 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[4] that uses the multiple GetP-$\mathsf{c}^{j}_{\downarrow}$'s to implement a single overall $\mathsf{GetP}_{\downarrow}$, so we can apply the algorithms for the first case.

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

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

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

(EnumCA-$\tau_{\mathcal{P}}$) Given a parameter sensibility threshold $\tau_{\mathcal{P}}$, EnumCA-$\tau_{\mathcal{P}}$ calls $\mathsf{GetP}_{\downarrow}$ repeatedly until it gets a parameter setting p with $\mathsf{SP}(p; p_0) \leqslant \tau_{\mathcal{P}}$, at which point EnumCA-$\tau_{\mathcal{P}}$ terminates and returns the parameter settings with the lowest relative strength seen so far.

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

(EnumCA$^{\text{weak}}$-$\tau_{\mathcal{R}}$) EnumCA$^{\text{weak}}$-$\tau_{\mathcal{R}}$ solves CA-$\tau_{\mathcal{R}}$ for the special case where a weak order on $\mathcal{P}$ is available (recall from Section 2.2.2 that CA-$\tau_{\mathcal{R}}$ requires only a partial order; that more general case will be discussed in Section 20). Given a result strength threshold $\tau_{\mathcal{R}}$, EnumCA$^{\text{weak}}$-$\tau_{\mathcal{R}}$ calls $\mathsf{GetP}_{\downarrow}$ repeatedly until it gets a qualified parameter setting p, i.e., with $\mathsf{SR}(q(p); r_0) < \tau_{\mathcal{R}}$. At this point, EnumCA$^{\text{weak}}$-$\tau_{\mathcal{R}}$ knows that p is an answer because unseen parameter settings must

[4] Note that this procedure differs from the analogous procedure for the second case above; see the end of Section 20 for detailed discussion.

have equal or lower sensibility. To get remaining answers (that tie with p in sensibility), $\overset{\text{weak}}{\mathsf{Enum}}\mathsf{CA}\text{-}\tau_{\mathcal{R}}$ continues until $\mathsf{GetP}_{\downarrow}$ yields a parameter setting with lower sensibility.

$\overset{\text{weak}}{\mathsf{Enum}}\mathsf{CA}\text{-}\tau_{\mathcal{R}}$ improves on $\mathsf{BaseCA}\text{-}\tau_{\mathcal{R}}$ because $\overset{\text{weak}}{\mathsf{Enum}}\mathsf{CA}\text{-}\tau_{\mathcal{R}}$ 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 $\mathsf{BaseCA}\text{-}\tau_{\mathcal{R}}$ 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 non-increasing 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 2-d skyline (Kung et al., 1975). Furthermore, $|A|$ grows monotonically, so we can terminate once $|A|$ reaches k. For details, see Algorithm 1.

EnumCA-po runs in $O(s(\mu_p + \mu_q))$ time, where $s \leqslant |\mathcal{P}|$ 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 $k \ll t$.

Enumerating Values in Each Dimension

Given the parameter setting p_0 of the original claim, suppose that for each dimension $\imath$ of the parameter space, a function $\mathsf{GetP}\text{-}\mathsf{d}^{\imath}_{\downarrow}(p_0)$ is available for returning values in dimension $\imath$ in order, such that $\mathsf{SP}(p; p_0)$ is monotonically non-increasing with respect to the ordinal number of p's value for dimension $\imath$ in the sequence returned by $\mathsf{GetP}\text{-}\mathsf{d}^{\imath}_{\downarrow}(p_0)$. Since it is possible for some combinations of single-dimensional values

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;
```

to be an invalid parameter setting, we also assume a Boolean function IsPValid$(p; p_0)$ that tests the validity of p. With these functions, we now show how to implement an overall GetP$_\downarrow$ by combining these multiple GetP-d$^i_\downarrow$'s.

A simple strategy is to obtain the list of values for each dimension i by repeatedly calling GetP-d$^i_\downarrow(p_0)$, 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 dimension 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 GetP-d$^i_\downarrow(p_0)$ 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 is initialized with an entry whose components are obtained by calling each GetP-d$^i_\downarrow(p_0)$ 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 $\mathsf{GetP}_{\downarrow}$. 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_i with the value that $\mathsf{GetP\text{-}d}^i_{\downarrow}(p_0)$ returns after v_i. 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;[5] cached values from $\mathsf{GetP\text{-}d}^i_{\downarrow}$ 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 dimensions, the time complexity can be higher than the baseline. However, this algorithm can stop execution early, enumerating only parameters in the "neighborhood" of p_0. Additional analysis in Section 3.2 will demonstrate this point analytically on a concrete problem instance.

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 $\jmath$, $\mathsf{GetP\text{-}c}^{\jmath}_{\downarrow}(p_0)$ is available for generating parameter settings in decreasing order according to $\jmath$ (again, we assume no ties for brevity; it

[5] 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 $\dim(\mathcal{P}) - 1$ dimensions is one-to-one.

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

For each dimension $\imath$, $V_\imath$ caches the values returned by $\mathsf{GetP\text{-}d}^{\imath}_{\downarrow}$ as a list. $V_\imath.\mathsf{len}()$ returns the length of the list; $V_\imath.\mathsf{get}(j)$, where $0 \leqslant j < V_\imath.\mathsf{len}()$, returns the j-th value; $V_\imath.\mathsf{append}(v)$ appends v to the list.

Candidate parameter settings are stored in a priority queue Q with sensibility as priority. More precisely, instead of storing the value for each dimension $\imath$, we store the index of this value in $V_\imath$. $Q.\mathsf{add}(e, s)$ adds entry e with priority s to Q; $Q.\mathsf{removeMax}()$ deletes the entry e with the highest priority s in Q, and returns (e, s); $Q.\mathsf{contains}(e)$ tests if Q contains entry e.

$d \leftarrow \dim(\mathcal{P})$;
$V_\imath \leftarrow \varnothing$ **foreach** $\imath \in [1, d]$;
def $\mathsf{nextV}_\imath(j)$ **begin**
  if $j = V_\imath.\mathsf{len}() - 1$ **then**
    $v \leftarrow \mathsf{GetP\text{-}d}^{\imath}_{\downarrow}(p_0)$;
    if $v = \perp$ **then return** $\perp$; ;
    $V_\imath.\mathsf{append}(v)$;
  return $j + 1$;
$Q \leftarrow \varnothing$;
def $\mathsf{p}(\langle j_1, j_2, \ldots, j_d \rangle)$ **begin**
  return $(V_1.\mathsf{get}(j_1), V_2.\mathsf{get}(j_2), \ldots, V_d.\mathsf{get}(j_d))$;
$Q.\mathsf{add}\big(\langle \mathsf{nextV}_1(-1), \ldots, \mathsf{nextV}_d(-1) \rangle, \mathsf{SP}(\mathsf{p}(\langle \underbrace{0, \ldots, 0}_{d} \rangle); p_0)\big)$;
while $Q \neq \varnothing$ **do**
  $(e, s) \leftarrow Q.\mathsf{removeMax}()$; *// remove the highest-sensibility entry*
  for $\imath \leftarrow 1$ **to** d **do** *// add successor e' along each dimension*
    $e' \leftarrow e$; $e'[\imath] \leftarrow \mathsf{nextV}_\imath(e[\imath])$;
    if $e'[\imath] \neq \perp$ **then**
      $Q.\mathsf{add}(e', \mathsf{SP}(\mathsf{p}(e'); p_0))$;
    if $e[\imath] > 0$ **then break**; ;
  if $\mathsf{IsPValid}(\mathsf{p}(e); p_0)$ **then yield** $\mathsf{p}(e)$; ;

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: $\forall \jmath$, $\mathsf{GetP\text{-}c}^{\jmath}_{\downarrow}(p_0)$ returns p_1 later than p_2. The algorithm for CA-$\tau_{\mathcal{R}}$ can be improved as follows.

(**$\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$**) $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ calls each $\mathsf{GetP\text{-}c}^{\jmath}_{\downarrow}(p_0)$ in a round-robin fashion, and stops as soon as it has encountered (and processed) a parameter setting that has been returned by all $\mathsf{GetP\text{-}c}^{\jmath}_{\downarrow}(p_0)$'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 w.r.t. each criterion.

As in BaseCA-$\tau_{\mathcal{R}}$, $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ maintains the maximal subset A of all qualified parameters settings seen so far. For each qualified parameter setting p (i.e., with $\mathsf{SR}(q(p); r_0) > \tau_{\mathcal{R}}$), $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ 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, e.g., range tree. Note that $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ 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 orthongonal range counting data structure (Chazelle, 1988), in the worst case, $\overset{\text{partial}}{\text{EnumCA}}$-$\tau_{\mathcal{R}}$ has time complexity $O(|\mathcal{P}|(\log^c t + \mu_p + \mu_q))$ and space complexity $O(t)$.[6] In practice, 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

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

case, using Fagin's Threshold Algorithm (Fagin et al., 2003), it is straightforward to implement $\mathsf{GetP}_{\downarrow}(p_0)$ by calling the $\mathsf{GetP\text{-}c}^{j}_{\downarrow}(p_0)$'s. With this $\mathsf{GetP}_{\downarrow}(p_0)$, we can then use $\mathsf{EnumCA}\text{-}\tau_{\mathcal{P}}$ and EnumCA-po (Section 2.3.2) 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 17, there is a fine but important distinction. Here, each $\mathsf{GetP\text{-}c}^{j}_{\downarrow}$ returns (full) parameter settings in $\mathcal{P}$. In contrast, each $\mathsf{GetP\text{-}d}^{i}_{\downarrow}$ returns only values for a single dimension i 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 $\mathsf{GetP\text{-}c}^{j}_{\downarrow}$'s, the same approach does not work for $\mathsf{GetP\text{-}d}^{i}_{\downarrow}$'s — the algorithm in Section 17 is needed.

2.3.3 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 3.2.2, 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* [7] 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: $\mathsf{OptP}_{\text{-}\infty}(r_0, \psi)$, for CA, returns $\arg\min_{p \in \psi} \mathsf{SR}(q(p); r_0)$, i.e., the parameter setting(s) in ψ with minimum result strength relative to r_0; $\mathsf{OptP}_0(r_0, \psi)$, for RE, returns $\arg\min_{p \in \psi} |\mathsf{SR}(q(p); r_0)|$, i.e., the parameter setting(s) in ψ whose result is closest to r_0.

Using DivP and $\mathsf{OptP}_{\text{-}\infty}$, we have the following algorithms for CA. The algorithms for RE are mostly identical, except that they use OptP_0 instead of $\mathsf{OptP}_{\text{-}\infty}$. 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 $\mathsf{DivP}(p_0, \tau_{\mathcal{P}})$ to divide the set of parameter settings above the sensibility threshold (relative to p_0) into a set of zones. Then, for each zone ψ, LocCA-$\tau_{\mathcal{P}}$ calls $\mathsf{OptP}(r_0, \psi)$ 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

[7] 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.

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. See Algorithm 3 for details.

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 3.2.2 and Section 3.3.2

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

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* $\langle \mathsf{key}, \mathsf{val} \rangle$ *pairs, sorted in decreasing* key *order. Methods* prepend$(\cdot)$ *and* removeLast$()$ *are self-explanatory. 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 $\tau \in (\check{s}_p, \tau^{\mathsf{top}})$ 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.

$A^{\mathsf{top}} \leftarrow \varnothing$; $\tau^{\mathsf{top}} \leftarrow \hat{s}_p$; $A^{\mathsf{bot}} \leftarrow \varnothing$;
$\tau \leftarrow$ initial guess in $(\check{s}_p, \hat{s}_p)$;
while $\tau - \check{s}_p > \epsilon$ **do**
  $A^{\mathsf{bot}} \leftarrow \varnothing$; $\gamma \leftarrow \tau$;
  while true do *// grow A^{bot} upwards*
    $(\gamma', \Delta) \leftarrow$ LocCA-$\tau_{\mathcal{P}}(\langle q, p_0, r_0 \rangle, \gamma)$;
    if $\Delta = \varnothing$ or $\gamma' > \tau^{\mathsf{top}}$ **then** *// A^{bot} merges into A^{top}; start new round*
      $A^{\mathsf{top}} \leftarrow A^{\mathsf{top}} \cup A^{\mathsf{bot}}$; $\tau^{\mathsf{top}} \leftarrow \tau$; **break**;
    A^{bot}.prepend$(\langle \gamma', \Delta \rangle)$;
    while $A^{\mathsf{bot}} \neq \varnothing$ and A^{top}.total$() + A^{\mathsf{bot}}$.totalXLast$() \geqslant k$ **do**
      A^{bot}.removeLast();
    $\gamma \leftarrow \gamma'$;
  if A^{top}.total$() \geqslant k$ **then break**; ;
  $\tau \leftarrow$ new guess in $(\check{s}_p, \tau)$;
return A^{top};

that would enable the above algorithms for claims generalizing our running examples.

3

Fact-Checking: Applications and Experiments

3.1 Introduction

In Chapter 2, we have introduced the QRS-based framework and defined within the framework a series of interesting and practical problems for finding counter-arguments of questionable claims and reverse-engineering vague claims. Meta algorithms following a "pay-as-you-go" architecture were proposed to solve these computational problems.

In this Chapter, we study two types of claims, namely *Window Aggregate Comparison* claims (Section 3.2) and *Time Series Similarity* claims (Section 3.3), generalized from the Giuliani's adoption claim (Example 1) and the Congressional voting correlation claim (Example 2), respectively.

Extensive experiments are performed to validate the ability of our framework in modeling fact-checking tasks and producing meaningful results (Section 3.4.1), as well as the efficiency of the algorithms (Section 3.4.2).

3.2 WAC: Window Aggregate Comparison Claims

Having described our modeling and algorithmic frameworks, we now show how to check a class of claims generalizing Giuliani's in Example 1. The generalized claim template can also be applied to other time series data. As an example, we show its application on the US monthly unemployment data in Section 3.4 (Section 3.4.1 details our choice of model parameters for that application).

3.2.1 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 $\sum_{i\in(t-w,t]} x_i$.[1] The *window aggregate comparison (WAC) query template* is the function

$$q(w,t,d) = \frac{\sum_{i\in(t-w,t]} x_i}{\sum_{i\in(t-d-w,t-d]} x_i}, \tag{3.1}$$

which compares two windows of the same length $w \in [1, n-1]$ ending at t and $t-d$, respectively. We call $t \in [w+1, n]$ the *current time* and $d \in [1, t-w]$ the *lead*. Hence, the parameter space $\mathcal{P}$ is the set of points in $\mathbb{N}^3$ 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 2.2.2, we define the result strength function as $\mathsf{SR}(r; r_0) = r/r_0 - 1$. (On the other hand, if the claim boasts a decrease, e.g., "*crime rate is dropping*," we would replace r/r_0 with r_0/r in $\mathsf{SR}(r; r_0)$.)

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

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):

$$\mathsf{SP}(p; p_0) \propto \mathsf{Nat}(p) \cdot \mathsf{Rel}(p; p_0). \tag{3.2}$$

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

First, consider naturalness. In general, for time series data, certain durations are more natural than others. For example, for monthly data, multiples of 12 (i.e., years) are more natural than multiples of 3 but not of 12 (i.e., quarters), who are in turn more natural than 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 a domain of time durations often has a periodic structure, we define naturalness for such a domain using a set of (usually a few, and often not disjoint) *levels* whose union is $\mathbb{N}$. Each level ℓ is specified by a pair (χ_ℓ, π_ℓ). Here, χ_ℓ is the naturalness score associated with level ℓ; $\pi_\ell \geqslant 1$ is an integral period that defines the domain 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 $\max\{\chi_\ell \mid v \in \mathbb{N}^{(\ell)}\}$; i.e., the maximum score that v is associated with.

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

$$\mathsf{Nat}(p) = \mathsf{Nat}_{\mathsf{win}}(w) \cdot \mathsf{Nat}_{\mathsf{lead}}(d), \tag{3.3}$$

where $\mathsf{Nat}_{\mathsf{win}}$ and $\mathsf{Nat}_{\mathsf{lead}}$ are naturalness scoring functions for the domains of w and d as discussed above. Specifically, for Giuliani's claim, we define naturalness for both domains using three levels $(1, 1)$, $(e, 4)$, $(e^2, 8)$. Here, periods 4 and 8 reflect

the natural term lengths of New York City mayors; the choice of e as bases is for convenience (when multiplied with a Gaussian relevance term). More sophisticated naturalness modeling is certainly possible, but we have found this definition to be adequate in our experiments.

Second, consider relevance. Generally speaking, the relevance of a parameter setting decreases with the magnitude of perturbation from the parameter setting of the original claim. For WAC, let $p_0 = (w_0, t_0, d_0)$ denote the original parameter setting. We define $\mathsf{Rel}(p; p_0)$ to be a 3-d normal distribution centered at p_0, i.e.,

$$\begin{aligned}
&\mathsf{Rel}(p; p_0) = \mathsf{Rel}_{\mathsf{win}}(w; w_0) \cdot \mathsf{Rel}_{\mathsf{tEnd}}(t; t_0) \cdot \mathsf{Rel}_{\mathsf{lead}}(d; d_0), \text{ where} \\
&\mathsf{Rel}_{\mathsf{win}}(w; w_0) = \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{win}}} \exp\left(-\frac{(w-w_0)^2}{2{\sigma_{\mathsf{win}}}^2}\right), \\
&\mathsf{Rel}_{\mathsf{tEnd}}(t; t_0) = \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{tEnd}}} \exp\left(-\frac{(t-t_0)^2}{2{\sigma_{\mathsf{tEnd}}}^2}\right), \\
&\mathsf{Rel}_{\mathsf{lead}}(d; d_0) = \frac{1}{\sqrt{2\pi}\sigma_{\mathsf{lead}}} \exp\left(-\frac{(d-d_0)^2}{2{\sigma_{\mathsf{lead}}}^2}\right)
\end{aligned} \tag{3.4}$$

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

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 2.1b illustrates the overall parameter sensibility function—the product of naturalness and relevance—for Giuliani's claim when fixing $w = 6$.

Fact-Checking Tasks The modeling above immediately enables the formulation of all problems in Section 2.2.2 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; uniqueness could be useful in seeing whether the adoption trend existed under other mayors' terms (had more historical data been available).

3.2.2 Algorithmic Building Blocks for WAC

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.

Ordered Enumeration of Parameters

Enabling ordered enumeration of parameters for WAC is straightforward. As discussed in Section 17, 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 $\mathsf{IsPValid}$ to test the validity of (w, t, d) combinations. Algorithm 2 in Section 17 can then combine these functions automatically to provide ordered enumeration of full parameter settings.

We show how to implement $\mathsf{GetP\text{-}d}_{\downarrow}^{w}(w_0)$ for the dimension of window length w, where w_0 is from the original claim's parameter setting. Recall from Section 3.2.1 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

Algorithm 4: GetP-$\mathsf{d}^{w}_{\downarrow}(w_0)$ 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 $\mathsf{add}(\cdot,\cdot,\cdot)$ and $\mathsf{removeMax}()$ 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 ;
6          else w' ← w + π_ℓ; ;
7      else
8          if w = w_0 then w' ← w_0 − (w_0 mod π_ℓ) ;
9          else w' ← w − π_ℓ; ;
10     if w' ∈ [1, n] then return w' ;
11     else return ⊥; ;
12 Q ← ∅;
13 foreach level ℓ do
14     foreach increase ∈ {true, false} do
15         w ← next_ℓ(w_0, increase); Q.add(w, contrib_ℓ(w), ℓ);
16 while Q ≠ ∅ do
17     (w, s, ℓ) ← Q.removeMax();
18     w' ← next_ℓ(w, (w > w_0));
19     if w' ≠ ⊥ then Q.add(w', contrib_ℓ(w'), ℓ); ;
20     yield w;
```

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.

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

To understand the complexity of ordered enumeration, we note that all parameter

settings above a given sensibility $\tau_{\mathcal{P}}$ fall within the ellipsoid centered at p_0 given by

$$(p - p_0)^T \Sigma^{-1} (p - p_0) = -(2 \ln \tau_{\mathcal{P}} + \ln |\Sigma| + 3 \ln(2\pi)). \tag{3.5}$$

See Figure 3.1 for an illustration (a slice of the bounding ellipsoid is outlined in red; ignore the "zones" for now).

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.

Let $\tilde{r}$ denote the length of the longest semi-principal axis (measure by the number of possible values on it) of this ellipsoid; we call $\tilde{r}$ the *interesting solution radius*. For CA-$\tau_{\mathcal{P}}$, $\tilde{r}$ is determined by the given $\tau_{\mathcal{P}}$. For CA-$\tau_{\mathcal{R}}$ (or CA-po), $\tilde{r}$ is determined by the "effective" $\tau_{\mathcal{P}}$, i.e., the sensibility of the answer (or the lowest sensibility in the answer set, respectively). The same analysis applies to the variants of RE. Using the results in Section 17, 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.

The Locus Approach

We now show how to enable an efficient locus approach to checking WAC claims, by defining functions DivP and OptP (Section 2.3.3). The subset of $\mathcal{P}$ above sensibility threshold $\tau_{\mathcal{P}}$ is a set of 3-d grid points. Roughly speaking, our approach "slices" this subset using planes perpendicular to the w axis, and computes the best answer within each slice.

Divide In more detail, the parameter space division function DivP works as follows. For each possible window length w and for each naturalness level ℓ in the domain of d specified by $(\chi_\ell^d, \pi_\ell^d)$, consider the uniform (t, d) grid in the plane shown in Figure 3.1. For a subset of grid 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 subgrids, with spacing of π_ℓ^d, and offsets $0, 1, \ldots, \pi_\ell^d - 1$ along the t-dimension.[2] Formally, for each $h \in [0, \pi_\ell^d)$, we define such a subgrid as a zone $\psi_{w,\ell,h}$ as follows:

$$t \leqslant n \ \wedge \ t - d \geqslant w; \tag{3.6}$$

$$d \bmod \pi_\ell^d = 0; \tag{3.7}$$

$$\left(\frac{t-t_0}{\sigma_t}\right)^2 + \left(\frac{d-d_0}{\sigma_d}\right)^2 < -\ln \tau, \text{ where } \tau = \frac{\tau_{\mathcal{P}}}{\mathsf{Nat}_{\mathsf{win}}(w) \cdot \mathsf{Rel}_{\mathsf{win}}(w; w_0) \cdot \chi_\ell^d}; \tag{3.8}$$

$$t \bmod \pi_\ell^d = h. \tag{3.9}$$

Here is some intuition of why a zone is defined as above. Consider a zone $\psi_{w,\ell,h}$. First, with w fixed, $q(w, t, d)$ can be written as y_t / y_{t-d} (y will be formally defined below). Given this decomposition of q, in order to maximize/minimize $q(w, t, d)$, we only need to maximize/minimize y_t and minimize/maximize y_{t-d}, which is a 1-d problem. However, not all combinations of (t, d) are valid according to the semantics of WAC claim. Constraint (3.9) ensures that for any t, values of d for valid (t, d) combinations are contiguous in $\psi_{w,\ell,h}$, which is convenient for optimization as shown below.

Optimize: CA Next, we show how to optimize each zone returned by DivP. For the problem of finding counterarguments, we define OptP_∞ for WAC (Algorithm 5) as follows. Note that each zone $\psi_{w,\ell,h}$ is associated with an equally-spaced subsequence

[2] This further division is for the convenience of OptP. With this division, each zone is associated with an equally-spaced sequence of time points, such that every pair of time points in this sequence satisfying Constraints (3.6) and (3.8) corresponds to a parameter setting in the zone.

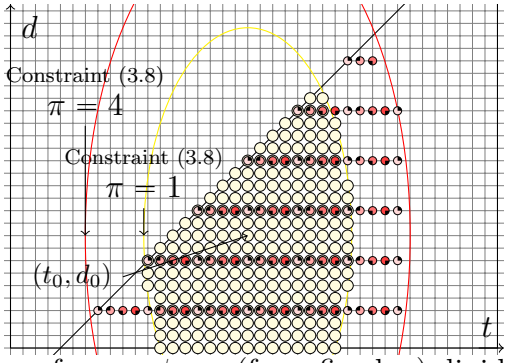


FIGURE 3.1: Illustration of zones $\psi_{w,\ell,h}$ (for a fixed w) dividing the parameter space of WAC claims.

of time points $I = \{i \mid i \bmod \pi_\ell^d = h \wedge i^{\min} \leqslant i \leqslant i^{\max}\}$, where $i^{\min}$ and $i^{\max}$ are derived from the constraints on t and d imposed by the zone. We compute the window aggregate result (with window length w) for each of these time points, obtaining a sequence $Y = \{y_i \mid i \in I\}$, where $y_i = \sum_{j \in (i-w,i]} x_j = \bar{x}_i - \bar{x}_{i-w}$. Recall that the $\bar{x}_i$'s are precomputed prefix sums, so computing Y takes $O(|Y|)$ time. Every (t, d) setting in zone $\psi_{w,\ell,h}$ corresponds to a pair of time 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 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).[3] We use the same notion of "interesting solution radius" $\tilde{r}$ introduced in Section 3.2.2 to analyze the complexity of the locus approach

[3] We actually use a simpler 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.

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

Lines 1, 5, and 6 optimize under Constraints (3.6), (3.7), (3.8), *and* (3.9)*; 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** $(i \leftarrow i^{\min}; i \leqslant i^{\max}; i \leftarrow i + \pi^d_\ell)$ **do** $y_i \leftarrow \bar{x}_i - \bar{x}_{i-w}$; ;

3 $\mathfrak{Y} \leftarrow$ a data structure for $\{y_i\}$ supporting range-maximum queries;

// Search the zone, one t value at a time:

4 $t^\star \leftarrow \perp$; $d^\star \leftarrow \perp$; $s^\star \leftarrow \infty$;

5 **foreach** $i_2 \in \{t \mid \exists d : (w,t,d) \in \psi_{w,\ell,h}\}$ **do**

6 $\quad i_1^{\min} \leftarrow \min\{i_1 - d \mid (w,i_2,d) \in \psi_{w,\ell,h}\}$; $i_1^{\max} \leftarrow \max\{i_1 - d \mid (w,i_2,d) \in \psi_{w,\ell,h}\}$;

7 $\quad (i_1, y_{i_1}) \leftarrow \mathfrak{Y}.\mathsf{range\text{-}max}([i_1^{\min}, i_1^{\max}])$;

8 $\quad s \leftarrow \frac{y_{i_2}}{y_{i_1}}/r_0 - 1$;

9 $\quad$ **if** $s < s^\star$ **then**

10 $\quad\quad t^\star \leftarrow i_2$; $d^\star \leftarrow i_2 - i_1$; $s^\star \leftarrow s$;

11 **return** $(w, t^\star, d^\star)$;

for WAC with the DivP and $\mathsf{OptP}_{\text{-}\infty}$ 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.

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 $\mathsf{OptP}_{\text{-}\infty}$, 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 $\mathsf{OptP}_{\text{-}\infty}$, 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 $\mathsf{OptP}_{\text{-}\infty}$. Recall that $\mathsf{OptP}_{\text{-}\infty}$ preprocesses Y, issues a query for each i_2 to find the best i_1 in the associated range, and then picks the overall best (i_1, i_2)

pair. Here, however, we find the overall best (i_1, i_2) using a sweep line procedure, which essentially considers the i_1 ranges associated with all valid i_2's in batch. To illustrate, let us map the sequence $Y = \{y_i \mid i \in I\}$ to a set of points $\{(i, y_i) \mid i \in I\}$ in 2-d as shown in Figure 3.2. 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 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 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 1-d 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.

3.3 TSS: Time Series Similarity Claims

We now turn to a class of claims generalizing the vote correlation claim in Example 2.

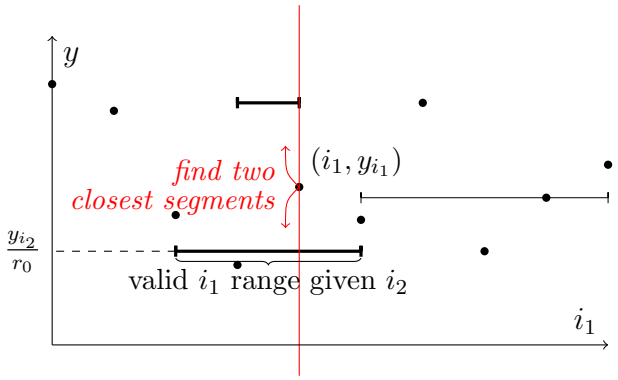


FIGURE 3.2: Illustration of the sweep line procedure used by OptP_0 for reverse-engineering WAC claims. Points corresponding to Y are drawn as black dots. Segments currently intersected by the (red) sweepline are drawn using thick lines.

3.3.1 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 $\text{sim}_T(X_u, X_v)$, 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 $\{x_{v,t} \in X_v \mid t \in T\}$, i.e., X_u and X_v restricted to the subset of timesteps in T. The *time series similarity (TSS) query template* is the function

$$q(u, v, a, b) = \text{sim}_{[a,b]}(X_u, X_v), \tag{3.10}$$

which compares *source entity* u against *target entity* v ($u \neq v$) over the time period $[a, b] \subseteq [1, n]$. Entity v is typically well recognizable to the audience, and serves as the target of comparison in order to claim something desirable about u.

For example, in the Marshall-Boehner claim (reverse-engineered) of Example 2, v is Boehner and u is Marshall; a corresponds to January 2007, and b corresponds to October 14, 2010, the time when the claim was made. Each vote is one of *yea*, *nay*, *present but not voting*, and *absent*. The similarity between two Representatives over a period is computed as the number of times they both voted *yea* or *nay*, divided by

the number of times neither is *absent*.

Result Strength and Parameter Sensibility We can investigate a TSS claim in multiple ways by parameter perturbation— changing the period of comparison, replacing the entities being compared, or both—which lead to multiple useful problem formulations. In many cases, it makes sense to perturb some instead of all parameters; doing so gives cleaner problem formulations and solutions that are easier to interpret. 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 Eq. (3.2).

In more detail, naturalness stems from a. In the vote correlation example, values of a that correspond to the beginning of some session of Congress are the most natural. As with the naturalness of durations discussed in Section 3.2.1, 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 domain values in level ℓ. Again, the naturalness score of a time point t is given by $\max\{\chi_\ell \mid \lambda_\ell(t) = \texttt{true}\}$; i.e., 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 Eq. (3.4), Section 3.2.1:

$$\mathsf{Rel}_{\mathsf{time}}(p; p_0) = (2\pi)^{-1}|\Sigma|^{-1/2} \exp\left(\frac{1}{2}(p - p_0)^T \Sigma^{-1}(p - p_0)\right), \quad \text{where } \Sigma = \mathrm{diag}({\sigma_{\mathsf{begin}}}^2, {\sigma_{\mathsf{end}}}^2). \tag{3.11}$$

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 definitions above, we obtain the three variants of the problem of finding counterarguments in Section 2.2.2, for perturbations of the comparison time period.

Finding Counterarguments by Perturbing Entities (TSS-CA$_{\boldsymbol{u}}$) 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 2).[4] 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

[4] It is also plausible to consider perturbing v alone; e.g., 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.

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: $u_1 \preceq^{u_0} u_2$ iff $\mathsf{Nat}_{\mathsf{src}}(u_1) \leqslant \mathsf{Nat}_{\mathsf{src}}(u_2)$ and $\mathsf{Nat}_{\mathsf{src}}(u_1; u_0) \leqslant \mathsf{Nat}_{\mathsf{src}}(u_2; u_0)$.

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 3.4 for how we obtain these annotations. We define:

$$\mathsf{Nat}_{\mathsf{src}}(u) = u.\texttt{rec}; \tag{3.12}$$

$$\mathsf{Nat}_{\mathsf{src}}(u; u_0) = u_0.\texttt{ide} - u.\texttt{ide}; \tag{3.13}$$

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.

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 above 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 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$_{ab}$:

$$\mathsf{SP}\left((v,a,b);(v_0,a_0,b_0)\right) = \mathsf{SP}_{\mathsf{tgt}}(v) \cdot \mathsf{Nat}_{\mathsf{begin}}(a) \cdot \mathsf{Rel}_{\mathsf{begin}}(a;a_0) \cdot \mathsf{Rel}_{\mathsf{end}}(b;b_0). \tag{3.14}$$

Given the nature of vague TSS claims, choices of v are often limited; such is the case for "Republican leaders" in Example 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 3.4.

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 2.2.2.

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 2.2.2, both fairness and robustness are useful in this case; uniqueness is not.

Second, considering other settings of u and v give 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 above, 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 multi-time series data sets, such as stock prices over time, and sports players' game-by-game performance stats. We do not detail the other applications in this dissertation.

3.3.2 Algorithmic Building Blocks for TSS

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 $\text{sim}_{[a,b]}(X_u, X_v)$ is a *distributive* or *algebraic aggregate* (Gray et al., 1996) over the set $\{\text{sim}_{[t,t]}(X_u, X_v) \mid t \in [a, b]\}$ of point-wise similarities—which is the case for vote correlation claims—we can do better. In this case, using an idea similar to that of prefix sums in Section 3.2.2, 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 $\text{sim}_{[a,b]}(X_u, X_v) = \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}$ 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.

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 3.2.2, i.e., by providing a function to enumerate values in each parameter dimension in order. The dimension of b contributes only to naturalness, so we simply enumerate b values in increasing distance from b_0. The dimension of a contributes to both naturalness and relevance, and can be handled in a way similar to the dimension of window length for WAC claims in Section 3.2.2 — 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 $a \leqslant b$.

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 (Eq. (3.12)) and ideology (Eq. (3.13)), respectively. It is straightforward to provide a function for enumerating entities by each criterion. Then, as discussed in Section 20, algorithm $\mathsf{Enum}^{\mathsf{partial}}\mathsf{CA}\text{-}\tau_{\mathcal{R}}$ immediately becomes applicable.

Finally, for TSS-RE, we extend the method for enumerating (a, b) described above for TSS-CA$_{ab}$ with the additional dimension of 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 Eq. (3.14)). For the Marshall-Boehner claim, this function simply enumerates all Representatives who can be regarded as "Republican leaders."

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 20, 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 2-d grid points. This subset is analogous to a slice of the 3-d grid points (with w fixed) in Section 20, but further division into zones is simpler in this case. For each naturalness level ℓ in the domain of a specified by $(\chi_\ell, \lambda_\ell)$, we let DivP return zone ψ_ℓ defined by the constraints below [5]:

$$1 \leqslant a \leqslant b \leqslant n; \tag{3.15}$$

$$\lambda_\ell(a) = \texttt{true}; \tag{3.16}$$

$$\left(\frac{a - a_0}{\sigma_a}\right)^2 + \left(\frac{b - b_0}{\sigma_b}\right)^2 < -\ln \frac{\tau_{\mathcal{P}}}{\chi_\ell}. \tag{3.17}$$

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

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 (3.15) and (3.17); the high-naturalness zone is a subset of the grid points with $a \in \mathbb{N}^{\mathsf{begin}}$.

$\mathsf{OptP}_{\text{-}\infty}$ 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 (3.15) and (3.17). Given a zone ψ_ℓ, for each value of a that is valid in ψ_ℓ, we compute

$$b_a^\star = \arg\min_{b \in J_a} \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}. \tag{3.18}$$

We describe a data structure for computing $b_a^\star$ for a given value of a.

Let $\mathcal{O} = \{o_i = (\bar{c}_i, \bar{s}_i) \mid 0 \leqslant i \leqslant n\}$ 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 3.3.2. For simplicity we assume that $n = 2^k$ for some integer $k \geqslant 0$. Since $\bar{c}_{i+1} \geqslant \bar{c}_i$ and $\bar{s}_{i+1} \geqslant \bar{s}_i$, 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

$$\mathcal{Q}(X, a) = \arg\min_{(\bar{c}_b, \bar{s}_b) \in X} \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}}. \tag{3.19}$$

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 $\mathcal{Q}(\mathcal{O}_a, a)$. Intuitively, the minimization objective is the slope of the line from o_{a-1} to o_b. Such a line must be tangent to the lower boundary of convex hull of X. This observation leads us to the following data structure supporting fast computation of lower convex hulls over different ranges of points in $\mathcal{O}$.

We construct a 1-d 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 (Andrew, 1979). Figure 3.3 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 (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 Berg et al. (2000) for details.

Given $a \in [1, n]$, we compute the set $\mathcal{C}_a$ of $O(\log n)$ nodes θ such that $\gamma_\theta \subseteq J_a$ but $\gamma_{p(\theta)} \nsubseteq J_a$, where $p(\theta)$ is the parent of θ. The nodes in $\mathcal{C}_a$ lie along two paths of $\mathbb{T}$. It is well known that $\mathcal{O}_a = \bigcup_{\theta \in \mathcal{C}_a} O^\theta$, so $\mathcal{Q}(\mathcal{O}_a, a) = \mathcal{Q}(\{\mathcal{Q}(O^\theta, a) \mid \theta \in \mathcal{C}_a\}, a)$. Fix a node $\theta \in \mathcal{C}_a$, since the line passing through 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 3.3 for an illustration. 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 $O(n \log n)$.

TSS-RE In this case, the space of (v, a, b) parameter settings has one more dimension v than the case of TSS-CA$_{ab}$ above. 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

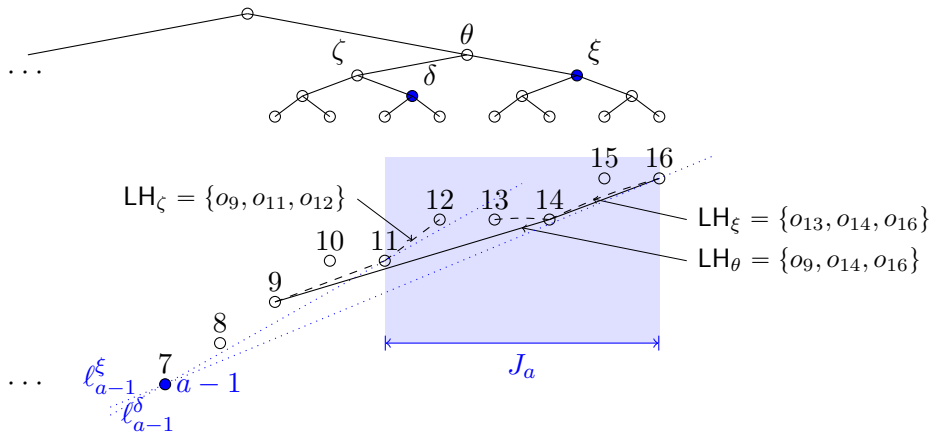


FIGURE 3.3: Illustration of the locus approach for TSS-CA$_{ab}$. The two dashed chains show LH_ζ and LH_ξ; the solid chain shows LH_θ. Suppose $a = 8$ and $J_a = [11, 16]$. Then $\mathcal{C}_a = \{\delta, \xi\}$ (note that $\mathsf{LH}_\delta = \{o_{11}, o_{12}\}$ is not labeled in the figure). Since ℓ^ξ_{a-1} has smaller slope than ℓ^δ_{a-1}, we have $b^\star_a = 16$ and $\mathfrak{Q}(\mathcal{O}_8, 8) = o_{16}$.

the case of TSS-CA$_{ab}$. More precisely, for each v with non-zero $\mathsf{SP}_{\mathsf{tgt}}(v)$ and for each naturalness level ℓ in the domain of a, we let DivP return zone $\psi_{v,\ell}$, which is defined by Constraints (3.15) and (3.16), as well as the following (which replaces Constraint (3.17)):

$$\left(\frac{a-a_0}{\sigma_a}\right)^2 + \left(\frac{b-b_0}{\sigma_b}\right)^2 < -\ln \frac{\tau_{\mathcal{P}}}{\mathsf{SP}_{\mathsf{tgt}}(v) \cdot \chi_\ell}. \tag{3.20}$$

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 (3.15) and (3.17), and let

$$\hat{b}_a = \arg\min_{b \in J_a} \left| \frac{\bar{s}_b - \bar{s}_{a-1}}{\bar{c}_b - \bar{c}_{a-1}} - r_0 \right|, \tag{3.21}$$

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 $\mathcal{O}_a^\top$ (resp. $\mathcal{O}_a^\perp$) be the subset of $\mathcal{O}_a$ of the points lying above (resp. below) ℓ_{a-1}, and let $b_a^\top$ (resp. $b_a^\perp$) be the index such that the line $\ell_{a-1}^\top$ (resp. $\ell_{a-1}^\perp$) passing through o_{a-1} and $o_{b_a^\top}$ (resp. $o_{b_a^\perp}$) has the smallest (resp. largest) slope among all lines passing through o_{a-1} and a point of $\mathcal{O}_a^\top$ (resp. $\mathcal{O}_a^\perp$). 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 3.4 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 semi-dynamic 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 slop 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 1-d 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 semi-dynamic 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 $\mathbb{T}$, 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-

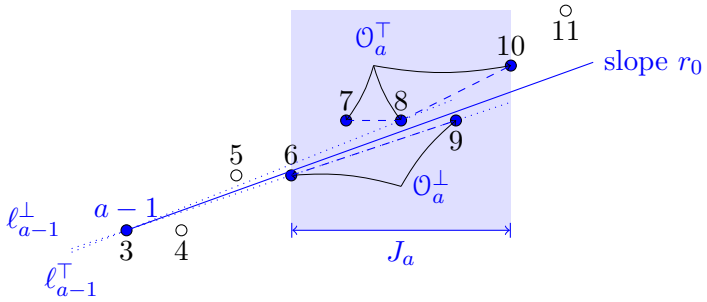


FIGURE 3.4: Illustration of the locus approach for TSS-RE. Suppose $a = 4$ and $J_a = [6, 10]$. The line of slope r_0 passes through o_{a-1}, and divides the points in range J_a into $\mathcal{O}_a^\top$ and $\mathcal{O}_a^\perp$. The lower and upper convex hulls of $\mathcal{O}_a^\top$ and $\mathcal{O}_a^\perp$ are shown as dashed chains. Dotted line $\ell_{a-1}^\top$ (resp. $\ell_{a-1}^\perp$) passes through o_{a-1} and $b_a^\top = o_8$ (resp. $b_a^\perp = o_6$). Because $\ell_{a-1}^\perp$ is closer to target slope, $\hat{b}_a = 6$.

CA_{ab}, i.e., $O(n \log n)$. Repeating this process for all m entities takes $O(mn \log n)$ time.

3.4 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 the New York City, we use the following datasets: ***UNEMP*** records the US monthly unemployment rate for 782 months from January 1948 to February 2013 [6]; ***AUTO*** contains daily auto accident statistics in Texas from 2003 to 2011 (with 3287 data points) [7]; ***VOTE*** contains 22.32 million votes cast by 12,572

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

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

members of the US Congress from May 1789 to April 2013. [8]

3.4.1 Proof of Concept

We first show the quality of results of solving the reverse-engineering and the counter-argument 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 3.2) and the UNEMP data, and then the TSS claim template on the VOTE data (as described in Section 3.3).

Giuliani's Adoption Claim

We first use our technique to reverse-engineer Giuliani's vague adoption claim in Example 1. Recall the model in Section 3.2.1. As discussed in Section 2.2.2, 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, i.e., Giuliani's first and second 4-year terms. This counterargument leads to 1% decrease in the adoption rate (as opposed to the big increase in the original claim), exposing the actual trend after Giuliani took office.

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

Marshall's Vote Correlation Claim

As another proof of concept, consider Marshall's vote correlation claim in Example 2. Recall the model in Section 3.3.1. 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. [9]

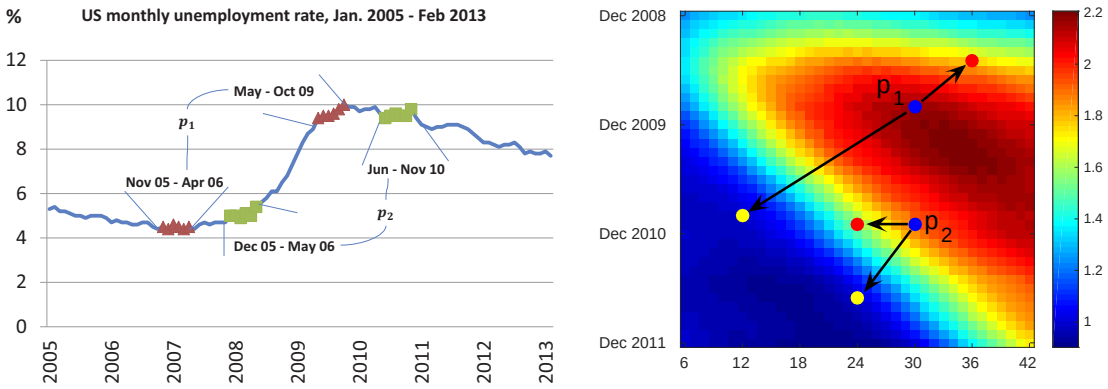
To reverse-engineer the original claim, we set up its context as follows. We let b_0 be the time when the claim was made (Oct. 2010), and a_0 be the start of the term in which the claim was made (Jan. 2010). We define the search space for the "Republican leader" as the five most recognizable Republican 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, (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%.

[9] `http://en.wikipedia.org/wiki/NOMINATE_(scaling_method)`

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.



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

FIGURE 3.5: 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.

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 (Fig. 3.5a). 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 3.5.

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).

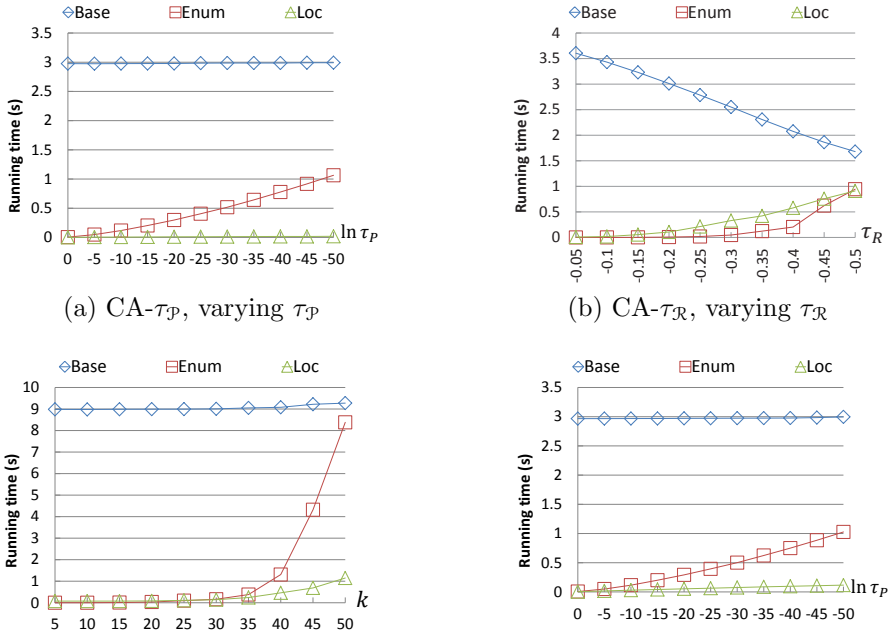
Both claims above point out increases in unemployment rate. A natural counterargument would point out a trend of decrease or lesser increase. We adopt the strength function $\mathsf{SR}(r; r_0) = r/r_0 - 1$ for $r_0 > 1$ from Section 2.2.1. For the sensibility function, we set $\sigma_w = 2.5$, $\sigma_t = 5$, and $\sigma_d = 10$. We use a 4-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).

From Figure 3.5a, 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 3.5b 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% w.r.t. r_2. To get an equally strong counterargument for Claim 1 (with a relative strength of -17.41% w.r.t. 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 $(6, \text{October 2010}, 12)$, while we only need to perturb p_2 to $(6, \text{July 2011}, 24)$. These counterarguments are shown as yellow dots pointed to by arrows in Figure 3.5b. The above results confirm our intuition that it is easier to find counterarguments for Claim 2, and imply that Claim 1 is more robust.

3.4.2 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). 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.



(a) CA-$\tau_{\mathcal{P}}$, varying $\tau_{\mathcal{P}}$

(b) CA-$\tau_{\mathcal{R}}$, varying $\tau_{\mathcal{R}}$

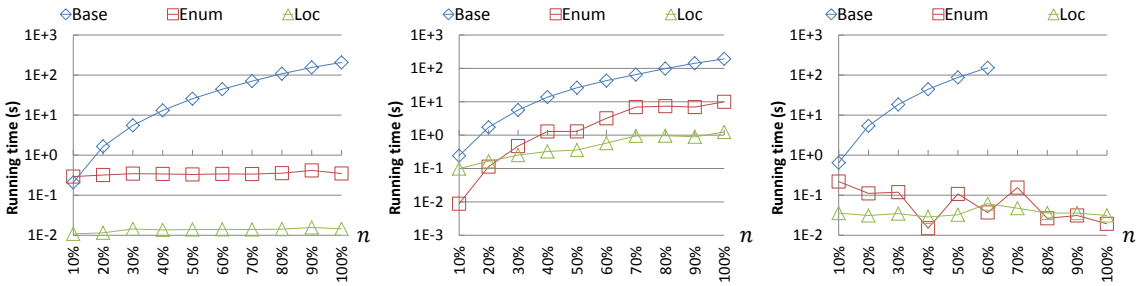
(c) CA-po, varying k

(d) RE-$\tau_{\mathcal{P}}$, varying $\tau_{\mathcal{P}}$

FIGURE 3.6: Running time of CA and RE algorithms for WAC claims on UNEMP.

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 3.2.2 and 3.3.2).

Varying $\tau_{\mathcal{P}}$ in CA-$\tau_{\mathcal{P}}$ for WAC on UNEMP Figure 3.6a shows the running times of the CA-$\tau_{\mathcal{P}}$ algorithms for WAC claims on UNEMP, as we vary the parameter sensibility threshold $\tau_{\mathcal{P}}$. Since Base always explores the entire parameter space $\mathcal{P}$, overall it is much slower than Enum and Loc. However, as $\tau_{\mathcal{P}}$ decreases, the region of $\mathcal{P}$ meeting this threshold becomes larger. Since the radius of this region is $O(|\ln \tau_{\mathcal{P}}|^{1/2})$, Enum needs to explore $O(|\ln \tau_{\mathcal{P}}|^{3/2})$ settings, which explains Enum's super-linear increase in running time in Figure 3.6a. Loc, with its powerful low-level building blocks, runs in time linear in $|\ln \tau_{\mathcal{P}}|$. This trend is difficult to see in the figure, as Loc remains fast even with very low sensibility thresholds.



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

FIGURE 3.7: Running time of CA algorithms for WAC claims on AUTO when varying data size.

Varying $\tau_{\mathcal{R}}$ in CA-$\tau_{\mathcal{R}}$ for WAC on UNEMP Figure 3.6b considers the CA-$\tau_{\mathcal{R}}$ problem for WAC claims on UNEMP, and compares the algorithms as we vary the result strength threshold $\tau_{\mathcal{R}}$. Here, as $\tau_{\mathcal{R}}$ decreases, we want counterarguments with results that deviate farther from the original claim 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 SP with fewer parameter settings that meet the result strength threshold. [10] 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-$\tau_{\mathcal{R}}$, Loc holds no advantage over Enum, which can be explained by the overhead of Loc's exponential search in this case.

Varying k in CA-po for WAC on UNEMP We now turn to CA-po, which returns the k Pareto-optimal counterarguments with highest sensibility. As explained in Section 2.2.2, this problem formulation is attractive because it avoids the sometimes tricky

[10] 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 3.6a. In practice, when query evaluation is more expensive, the opposite may hold.

task of choosing thresholds for CA-$\tau_{\mathfrak{P}}$ and CA-$\tau_{\mathfrak{R}}$. Figure 3.6c shows the running time of the three algorithms for WAC claims on UNEMP when we vary k. Enum and Loc show comparable performance up to $k = 35$. After that, the running time of Enum increases rapidly, and approaches that of Base. On the other hand, the running time of Loc shows a much slower increase and remains much faster than Base for all k values tested.

Varying $\tau_{\mathfrak{P}}$ in RE-$\tau_{\mathfrak{P}}$ for WAC on UNEMP For RE-$\tau_{\mathfrak{P}}$, the problem of reverse-engineering, Figure 3.6d compares the three algorithms for WAC claims on UNEMP. As we decrease the parameter sensibility threshold $\tau_{\mathfrak{P}}$, we observe the same trend as in Figure 3.6a for CA-$\tau_{\mathfrak{P}}$: Base is the slowest, while Loc scales better than Enum in the size of the high-sensibility region of the parameter space. Note that Loc for RE-$\tau_{\mathfrak{P}}$ in Figure 3.6d is slightly slower than Loc for CA-$\tau_{\mathfrak{P}}$ in Figure 3.6a, because of the more expensive building block (OptP_0 vs. $\mathsf{OptP}_{-\infty}$ in Section 20).

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 3.7, we show the results on the three variants of the problem of finding counterarguments—CA-$\tau_{\mathfrak{P}}$, CA-$\tau_{\mathfrak{R}}$, 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-$\tau_{\mathfrak{R}}$ (Figure 3.7b), 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 required strength threshold. For CA-$\tau_{\mathfrak{P}}$ (Figure 3.7a) and CA-po (Figure 3.7c), 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 3.8 compares the three algorithms for TSS-CA$_{ab}$, i.e., fixing two voters and perturbing the time period $[a, b]$ to find counterarguments that show lower vote correlation than originally claimed. Here, we consider the CA-$\tau_{\mathcal{P}}$ variant of the problem, and decrease the parameter sensibility threshold $\tau_{\mathcal{P}}$ (thereby enlarging the region of $\mathcal{P}$ meeting this threshold). We observe trends similar to those in Figures 3.6a and 3.6d 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 3-d for WAC but only 2-d here. Hence, Enum and Loc fare better here with an increasing search space.

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$_u$, i.e., 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 3.3.1, 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, $\mathsf{Enum}^{\mathsf{partial}}$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

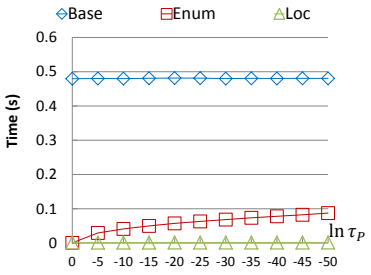


FIGURE 3.8: Running time of CA-$\tau_{\mathcal{P}}$ algorithms for TSS-CA$_{ab}$ on VOTE, varying $\tau_{\mathcal{P}}$.

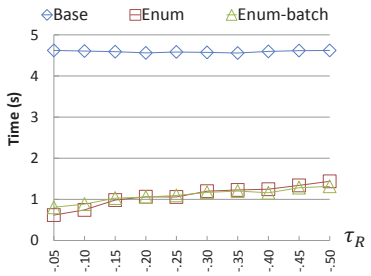


FIGURE 3.9: Running time of CA-$\tau_{\mathcal{R}}$ algorithms for TSS-CA$_u$ on VOTE, varying $\tau_{\mathcal{R}}$.

each Representative. Therefore, for Base, we computed all vote correlations in a single SQL query; for Enum, we tested both batched (with 5 correlation computations per SQL query) and unbatched versions.

Figure 3.9 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 3.9) 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.

3.5 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 3.6) includes investigating sampling and approximation, and extending our model to consider parameter and data perturbations jointly. For example, sampling and approximation are effective for large parameter spaces and data sizes. All algorithms in this chapter are exact; developing faster approximation algorithms would be a natural next step. Computing various claim quality measures from the QRS, a subject that this dissertation does not focus on, is clearly amenable to sampling. Finally, we are 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.

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.

The reverse-engineering problem is related to recent work on *query by output* (Tran et al., 2009) and *view synthesis* (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 above 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 towards the end of Section 2.1, fact-checking in general requires a repertoire of techniques including but not limited to the computational ones presented in this disseration, such as how to find datasets relevant to given claims, 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., 2006, 2007; 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.

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); 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.

3.6 Conclusion and Future Work

In this chapter, 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 this dissertation. There are several lines of work underway, including efficient computation of quality measures, approximation algorithms,

unified modeling of parameter and data changes, and going from fact-checking to lead-finding. Specialized building blocks for many other claim types remain to be discovered. Besides the interesting problems at the back-end, we are also working on making our techniques easier to apply. Along this line, we are investigating a learning-based approach that rely on user feedback to help specify functions SR and SP. The culmination of this work will be an end-to-end system to empower journalists and the public in combating the "lies, d—ed lies, and statistics" that permeate our public life today. To that end, we also need advances in complementary research areas such as source identification, data integration, data cleansing, natural language querying, and crowdsourcing.

4

Lead-Finding

4.1 Introduction

In Chapter 2, we have proposed a general QRS-based framework, for analyzing claims by treating a claim based on structured data as a query with parameters. We have modeled two classes of journalistic tasks, namely reverse-engineering vague claims and finding counter-arguments for questional claims, as bi-criteria optimization problems.

In this chapter, we shift the focus to the source of claims—we study the problem of finding "high-quality" leads given data and a claim template. We show that the techniques we use to solve this lead-finding task can be applied to the fact-checking problems as well.

Motivation

The bi-criteria optimization problems defined in Section 2.2.2 have served us well as an intuitive and convenient way of arguing against questionable claims, or reverse engineering vague ones. However, there are two aspects of quality this formulation cannot provide.

Let's revisit Giuliani's adoption claim (Example 1).

Example 3 (**Revisiting Giuliani's Adoption Claim**). In Chapter 1, we have countered Giuliani's adoption claim by saying that

(CA-1) Comparing Giuliani's first and second 4-year terms, i.e., 1994-1997 and 1998-2001, leads to 1% decrease.

This was the most sensible counter-argument found by CA-po. The next three down the list are

(CA-2) Comparing the two 4-year windows of 1995-1998 and 1999-2002, leads to 19% decrease.

(CA-3) Comparing the two 2-year windows of 1996-1997 and 2000-2001, leads to 24% decrease.

(CA-4) Comparing the last year of Giuliani's first and second half of his term, i.e., 1997 and 2001, leads to 32% decrease.

If we try to argue against CA-1, the most sensible counter-argument is

(CA-5) Comparing second half of Giuliani's term against the 4 years preceeding his term, the adoption in NYC increased by 99.34%.

Interestingly, CA-1 is also the most sensible counter-argument to CA-5. So is either or both of them nitpicking?

Looking at Example 3, the first problem we see is that the list of counter-arguments are not too different from each other. The solution to bi-criteria optimization problems are maximal points of the 2d point set $\{(\mathsf{SP}(p;p_0),\mathsf{SR}(q(p);r_0))\}_{p\in\mathcal{P}}$ (or $\{(\mathsf{SP}(p;p_0),|\mathsf{SR}(q(p);r_0)|)\}_{p\in\mathcal{P}}$ for reverse-engineering). Spatial structure of the parameter space is not considered. The consequence is that some areas of the parameter space have contain a lot of solution points, while other areas are covered by none. In other words, diversity of solution points in the parameter space cannot be provided by bi-criteria optimization formulations. In the context of lead-finding from data,

while we are still intereted in claims/parameters leading to extremal response values, the contraint on context, or sensibility, is much looser, sometimes even non-existent. More attention should be paid to spatial diversity of solution points to cover different areas of the parameter space (or a portion of it).

Secondly, the solution to bi-criteria optimization problems are optimized w.r.t. only the given claim. It is hard to tell if a counter-argument found in this way is a good claim by itself, for example, when we see two claims CC-1 and CC-5 countering each other in Example 3. We can verify a given claim's quality, say using quality measures such as *fairness* an d *robustness*, but how do we find high-quality ones in the first place?

Motivated by the application in finding high quality leads, and with the demand for spatial diversity, in this chapter, we focus on addressing the following question.

> *Given the parameter space (or a portion of it) of a claim template defined on some data set, and a budget of k points to choose from the space to represent the high-quality areas of the surface, which k points should we choose?*

The problem we are interested in is naturally related to top-k queries with diversification (Carbonell and Goldstein, 1998; Qin et al., 2012) and weighted clustering (MacQueen, 1967; Lloyd, 1982; Ackerman et al., 2012; Chen and Wang, 2011). At a high level, existing methods for diversified top-k do not consider the relationship between the k chosen points and the rest of the problem space, thus cannot guarantee that the chosen points represent high-quality areas. On the other hand, weighted clustering methods such as weighted k-means focus on clustering rather than choosing high-quality representatives. Detailed comparison will be presented in Section 4.6.1.

To outline this chapter, in Section 4.2, we will formally define the k-REPS problem for finding k representative points, accounting for various aspects of "high qual-

ity". In Section 4.3, we discuss how to apply the k-REPS problem for finding high-quality leads within the QRS-based framework. A concrete example of applying the k-REPS problem is shown in Section 4.4 using the *Time Series Similarity (TSS) claim* template discussed in Section 3.3 on the *US Congressional Voting* data. In Section 4.5, we propose algorithms for efficiently solving the k-REPS problem. In Section 4.6, proof-of-concept experiments are conducted to show the difference between the results found by our k-REPS problem and existing related approaches. We also perform extensive experiments to verify the efficiency of the proposed algorithms.

4.2 The k-REPS problem

Consider a surface given as a non-negative-valued real function g defined on a bounded domain $\mathcal{D}$, our goal is to select k high-valued points [1] from $\mathcal{D}$ to represent high-valued regions of the surface.

Given a representative s, let potential function $\phi_s : \mathcal{D} \mapsto \mathbb{R}_{\geqslant 0}$ denote the *impact* of s on points of $\mathcal{D}$. The impact of s is proportional to its own quality, i.e., $g(s)$. The impact of s on $p \in \mathcal{D}$ is discounted by the distance between s and p, denoted by a kernel function $\mathcal{K}(\cdot; s) : \mathcal{D} \mapsto \mathbb{R}_{\geqslant 0}$ centered at s. Combining the two aspects of impact above, we define the impact of representative s on $p \in \mathcal{D}$ as

$$\phi_s(p) = g(s) \cdot \mathcal{K}(p; s). \tag{4.1}$$

Given a set $S \subseteq \mathcal{D}$ of representatives, let function $\Phi_S : \mathcal{D} \mapsto \mathbb{R}_{\geqslant 0}$ denote the impact of S on points of $\mathcal{D}$. We define Φ_S as

$$\Phi_S(p) = \max_{s \in S} \phi_s(p). \tag{4.2}$$

[1] Without loss of generality, suppose higher surface value corresponds to higher quality/interestingness of individual point.

Φ_S is the upper envelope of ϕ_S. By defining $\Phi_S(p)$ as the maximum impact of individual representatives on p, we are effectively partitioning the domain $\mathcal{D}$ into "max-impact" regions of the representatives of S.

The objective is to maximize the total impact of a set of k representatives $S \subseteq \mathcal{D}$ on the entire problem domain $\mathcal{D}$, where the impact of S on $p \in \mathcal{D}$ is weighed by the quality of p, i.e., $g(p)$, in order to capture the intuition that points of higher quality demand better coverage by S.

The k-representative (k-REPS) problem is formally defined as follows.

Definition 1 (*k-representative (k-REPS) problem*)**.** Given a surface function $g : \mathcal{D} \mapsto \mathbb{R}^+$, and a natural number $k > 0$, find a set $S^* \subseteq \mathcal{D}$ defined as

$$S^* \triangleq \arg \max_{S \subseteq \mathcal{D}, |S|=k} \mathcal{G}_{\mathcal{D}}(S), \tag{4.3}$$

where

$$\mathcal{G}_{\mathcal{D}}(S) \triangleq \sum_{p \in \mathcal{D}} g(p) \cdot \Phi_S(p) \tag{4.4}$$

$$= \sum_{p \in \mathcal{D}} g(p) \cdot \max_{s \in S} \phi_s(p) \tag{4.5}$$

$$= \sum_{p \in \mathcal{D}} g(p) \cdot \max_{s \in S} g(s) \cdot \mathcal{K}(p; s). \tag{4.6}$$

Diversification kernel In Eqn. 4.1, $\mathcal{K}$ controls the discount of s's impact on p as the distance $\delta(p, s)$ between p and s increases, and is called the *diversification kernel*.

For the rest of this chapter, suppose the problem domain $\mathcal{D}$ is d-dimensional, we use the (unnormalized) d-dimensional Guassian kernel for $\mathcal{K}$, i.e.,

$$\mathcal{K}(p; s) = \exp\left\{-\frac{C}{2}\delta^2(p, s)\right\} = \exp\left\{-\frac{C}{2}(p - s)^T \Sigma^{-1}(p - s)\right\}. \tag{4.7}$$

In the definition of $\mathcal{K}$ above, Σ is a $d \times d$ co-variance matrix. A constant $C \geqslant 0$ is used to scale the distance function δ, hence controlling the degree of diversification. To see how a smaller value of C provides more diversification, consider the two extreme cases.

- $C = 0$. We have $\mathcal{K}(p; s) = 1$. The impact of a representative s on p does not decrease as p goes away from s. In this case, $\phi_s(p) = g(s) \cdot \mathcal{K}(p; s) = g(s)$. It suffices to have a singleton representative set $S^* = \{\arg\max_{p \in \mathcal{D}} g(p)\}$ to best represent all points of $\mathcal{D}$. In this case $\mathcal{G}_{\mathcal{D}}(S^*) = \max_{p \in \mathcal{D}} g(p) \cdot \sum_{p \in \mathcal{D}} g(p)$.

- $C \to +\infty$. $\mathcal{K}(p; s) = 1$ if $p = s$, and 0, otherwise. In this case, a representative has no impact on any other point of $\mathcal{D}$. Consider the objective function—$\mathcal{G}_{\mathcal{D}}(S) = \sum_{s \in S} g(s)$. The optimal solution S^* to the k-REPS problem would consist of the k points of $\mathcal{D}$ with the highest g-values, regardless of spatial diversity, hence no diversification at all.

In general, two chosen representatives that are "too close" to each other provides redundant coverage and should be avoided for the purpose of maximizing $\mathcal{G}$. Adjusting the value of C effectively adjust the threshold of two points deemed to "too close", and trades off utility versus diversity. The distinction between our formulation and a graph-based diversified top-k approach (Qin et al., 2012), as we shall see in Section 4.6.1, is that the graph-based approach uses a hard threshold preventing two close points from both being included in the solution set, while our design of the optimization objective $\mathcal{G}$ automatically penalizes such choices.

Remark (Alternative choice of $\mathcal{K}$)**.** Alternatively, instantiating the kernel function $\mathcal{K}(p; s)$ as $\frac{1}{\delta(p,s)+C}$ would yield the reciprocal kernel. [2] In this case, focusing on the

[2] The constant term C is added to the distance function $\delta(p, s)$ as a regulerization term, so as not to overly reward p's that are too close to s, and to avoid division-by-zero when $p = s$.

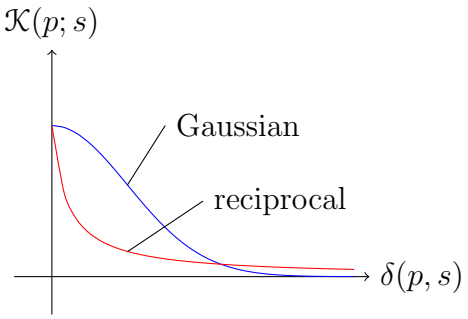


FIGURE 4.1: Comparing Gaussian kernel and reciprocal kernel for diversification

contribution of a single point $p \in \mathcal{D}$ to the objective function $\mathcal{G}$, $g(p) \cdot \Phi_S(p)$ can be interpreted as the potential energy between p and s if we view a point p as a charged particle with charge $g(p)$. However, compared to Gaussian kernel, which gives a smooth decrease of impact by s in close neighborhood of s, the reciprocal kernel yields a sharp decrease around s (Figure 4.1). For this reason, we find the reciprocal kernel less favorable than the Gaussian kernel for lead-finding purpose.

Distance function δ In the definition of the Gaussian diversification kernel (Eqn. 4.1), we adopted the *Mahalanobis distance* for the choice of the distance function δ, i.e.,

$$\delta^2(p,s) = (p-s)^T \Sigma^{-1} (p-s). \tag{4.8}$$

When the covariance matrix Σ is diagonal, it reduces to the scaled Euclidean distance.

Another sensible option for the distance function δ is the *geodesic distance*. While Mahalanobis distance measures captures the distance between two points in the parameter space, geodesic distance, which measures the length of the shortest path between two points on a multi-dimensional surfaces, also takes into account the surface function g. Here is a scenario in which geodesic distance can be a better choice than Mahalanobis distance.

Consider two smaller high-valued regions close to each other in terms of Mahalanobis distance, versus a larger region of similar elevation. Without accounting for the g function in δ, the k-REPS problem will inevitably pick at least two representatives from the larger region if two representatives were to be chosen to cover the two smaller regions, one each. In contrast, for two high-valued points from the two respective high-valued regions, geodesic distance would deem them much further than two points from the larger region, of equal Mahalanobis distance.

However, from the efficiency perspective, geodesic distance is much more expensive to compute than Mahalanobis distance—computing geodesic distance takes $\Omega(|\mathcal{D}|)$ time in the worse case (Kimmel and adn Alfred M Bruckstein, 1995; Kimmel and Sethian, 1998), while computing Mahalanobis distance takes $O(1)$ time.

Non-diversification Subspace As we have been discussing so far, one key aspect of the quality of the solution set to the k-REPS problem is spatial diversity. However, there are situations where diversification is not needed along some attribute, or in some subspace. For example, when categorical-valued attribute is present in the parameter space it is less straightforward to measure the distance between values, and undesirable to diversify the result along such attribute.

Let the problem domain $\mathcal{D}$ be a finite subset of a d-dimensional parameter space $\mathcal{P}$, consisting of two orthogonal subspaces — $\mathcal{P}_{\mathtt{D}}$, where diversification is desired, and $\mathcal{P}_{\mathtt{ND}}$, where no diversification is needed, i.e.,

$$\mathcal{D} \subseteq \mathcal{P} \subseteq \mathcal{P}_{\mathtt{ND}} \times \mathcal{P}_{\mathtt{D}},$$

$$\dim(\mathcal{P}_{\mathtt{D}}) + \dim(\mathcal{P}_{\mathtt{D}}) = d.$$

For example, as we'll show in Section 4.4, for *Time Series Similarity (TSS) claim* on the *US Congressional Voting* data, it only makes sense to diversify the time interval of comparison, not the the entity being compared with. There, we have

$\mathcal{P}_{\mathtt{D}} = \mathbb{Z}^2$ for the starting and ending times for comparison, and $\mathcal{P}_{\mathtt{ND}} = V$ for the target entity/voter being compared.

For each $p \in \mathcal{D}$, let $\pi_{\mathtt{D}}(p)$ and $\pi_{\mathtt{ND}}(p)$ denote the projections of p on $\mathcal{P}_{\mathtt{D}}$ and $\mathcal{P}_{\mathtt{ND}}$, respectively. To avoid diversification in subspace $\mathcal{P}_{\mathtt{ND}}$, we define the diversification kernel $\mathcal{K}$ such that

$$\mathcal{K}(p; s) > 0 \Rightarrow \pi_{\mathtt{ND}}(p) = \pi_{\mathtt{ND}}(s), \tag{4.9}$$

In other words, diversification is considered only among points of $\mathcal{D}$ with the same projection on $\mathcal{P}_{\mathtt{ND}}$. This can be achieved by defining the co-variance matrix Σ of the Mahalanobis distance function δ (Eqn. 4.8) such that

- $\Sigma_{AB} = 0$ if attribute A or B belongs to $\mathcal{P}_{\mathtt{ND}}$, and $A \neq B$.
- $\Sigma_{AA} = +\infty$ if attribute A belongs to $\mathcal{P}_{\mathtt{ND}}$.

We discuss later in Section 4.5.2 the algorithmic implication of the way we define $\mathcal{K}$ for non-diversification subspace $\mathcal{P}_{\mathtt{ND}}$.

4.3 Deriving the Surface g.

The definition of the k-REPS problem requires a non-negative-valued real surface function g in which higher value represents higher point quality. This is not necessarily true, for example, for the response surface function $q : \mathcal{P} \mapsto \mathcal{R}$ (Section 2.2.1), when $\mathcal{R} = \mathbb{R}$.

One possible way to normalize an bounded real-valued function $h : \mathcal{D} \mapsto \mathbb{R}$ to range $[0, a]$ $(a > 0)$ is to apply a linear transformation as follows, which preserves relative value differences.

$$\tilde{h}(x) = a \cdot \frac{h(x) - \min_y h(y)}{\max_y h(y) - \min_y h(y)}. \tag{4.10}$$

Alternatively, a purely rank based normalization of h can be defined as follows.

$$\tilde{h}(x) = a \cdot \frac{|\{y \in \mathcal{D} \mid h(y) \leqslant h(x)\}|}{|\mathcal{D}|}. \tag{4.11}$$

Another option for normalizing h is to use the logistic/sigmoid function.

$$\tilde{h}(x) = L(h(x); \sigma_h^{-1}, \bar{h}), \tag{4.12}$$

where

$$L(x; b, x_0) = \frac{a}{1 + e^{-b(x-x_0)}}. \tag{4.13}$$

In Eqn. 4.12 above, $\bar{h}$ and σ_h are the mean and standard deviation of $h(x)$, respectively, on $\mathcal{D}$, i.e., [3]

$$\bar{h} = \frac{1}{|\mathcal{D}|} \cdot \sum_{x \in \mathcal{D}} h(x), \tag{4.14}$$

$$\sigma_h = \left(\frac{1}{|\mathcal{D}|} \cdot \sum_{x \in \mathcal{D}} (h(x) - \bar{x}) \right)^{1/2}. \tag{4.15}$$

Normalization of h using the logistic function stretches the relative value difference near $\bar{h}$, and reduces the relative difference within high values and low values. Compared to linear transformation, a g function derived from logistic normalization favors high-quality areas than representatives of extremal h-values.

To promote low-valued points, all three methods can be reversed as $\tilde{h}' = a - \tilde{h}$.

Choice of normalization method depends on the task, and the distribution of values of h. For example, suppose the second highest h-value is much smaller than the highest, a linear combination of the linear transformation and the rank-based

[3] Definition of $\bar{h}$ and σ_h can be extended to continuous domain $\mathcal{D}$.

normalization can be used to shrink the gap, or the logistic normalization can be used to serve the same purpose.

In lead-finding, interesting leads come from extremal response value. The response function q can be normalized to derive g, i.e.,

$$g(p) = \tilde{q}(p). \tag{4.16}$$

Similarly, in finding counter-arguments for a given claim/parameter p_0, the relative result strength function SR can be normalized and used as the input function g to the k-REPS problem, i.e.,

$$g(p) = \tilde{\mathsf{SR}}(q(p); q(p_0)). \tag{4.17}$$

For reverse-engineering a vague claim of response r_0, g can be defined as

$$g(p) = \tilde{h}(p), \text{ where } h(p) = |\mathsf{SR}(q(p); r_0|). \tag{4.18}$$

The target surface g of the k-REPS problem can also be designed to account for other aspects of a claim's quality, e.g. fairness, robustness, as a linear combination of the multiple (normalized) quality measures. For example, g can be defined as a linear combination of (normalized) response function q and (normalized) fairness function $\mu(p) = \sum_{p' \in \mathcal{P}} \mathsf{SP}(p'; p) \cdot q(p')$ (note the difference from the uniqueness measure defined in Section 2.2.2) as

$$g(p) = \lambda \cdot \tilde{q}(p) + (1 - \lambda) \cdot \tilde{\mu}(p), \text{ for } p \in \mathcal{D}. \tag{4.19}$$

Here, $\lambda \in [0, 1]$ can be considered as a parameter of user skepticism - given a claim p, the user trust in p itself with probability λ, and perturb it according to pmf $\mathsf{Rel}(\cdot; p)$ with probability $1 - \lambda$.

Replacing μ with some other quality measures, e.g. robustness, would lead to different interpretations, but the same method applies.

4.4 Modeling Example: Time Series Similarity Claim

In this Section, we show a concrete example of applying the k-REPS problem on the *Time Series Similarity (TSS) claim* discussed in the Section 3.3 on the *US Congressional Voting* data.

As described in Section 3.3, the US Congressional Voting data consists of information about m entities (House Representative) identified as $1, 2, \ldots, m$. Each Representative u is associated with a time series $X_u = \{x_{u,1}, x_{u,2}, \ldots, x_{u,n}\}$, representing his/her votes on bills $1, 2, ..., n$ in chronicle order. A TSS claim on such a multi-time-series data set is parameterized by (i) the *source entity* u and the *target entity* v of comparison, and (ii) the time interval, represented by a $[a, b] \subseteq [1, n]$, over which the similarity is evaluated.

The full 4-dimensional parameter space of a TSS claim can be written as a finite subset of the product of two subspaces — a 2-dimensional subspace V^2 for the entities of comparison, and a 2-dimensional subspace $\mathbb{Z}^2$ for the (discrete) starting and ending times of comparison. Typically, the source Representative u is fixed, and a claim states the voting correlation between u and another notable Representative v over certain period of time, which leaves three parameters open for perturbation, i.e., v, a, and b.

Formalizing the Problem The k-representative lead-finding problem for TSS claims on the Congressional Voting data can be formalized as follows.

For $p = (u, v, a, b)$,

$$q(p) = \mathrm{sim}_{[a,b]}(X_u, X_v).$$

The value of $q(p)$ ranges between 0 and 1 if defined.

Fix the source entity u in p. For $p' = (u, v', a', b')$, $\mathsf{Rel}(p'; p) > 0$ *iff* $v = v'$ (part of the fairness function μ).

The surface function g required by the k-REPS problem is defined as a linear combination of a parameter p's response and its fairness, weighted by the target entity v's (normalized) recognizability and ideology. Consider $\mathcal{D}$ being a subset of the full 4D parameter space $\mathcal{P}$ with identical value of the source entity. Formally, for $p = (u, v, a, b) \in \mathcal{D}$,

$$g(p) = (\lambda \cdot q(p) + (1 - \lambda) \cdot \mu(p)) \cdot \texttt{rec}(v) \cdot \texttt{ide}(v).$$

In the definition of the g function above, no normalization is needed on q or μ as the range of q (thus μ as well) is already restricted to $[0, 1]$ according to similarity definition. The recognizability and ideology functions are normalized per discussion in Section 4.3.

The ideology score of Representatives are obtained using the well-known *NOMINATE* method. [4] Representatives with lowest ideology scores are considered most liberal according to their voting behavior, and vice versa.

The recognizability score of a legislator is defined as the number of appearances on Sunday news shows. Details can be found at `http://media.cq.com/facetime`.

Further adjustment to $g(p)$ is needed depending on the purpose. For example, in order to paint u as conservative, one can claim u has high/low correlation with some v with conservative/liberal ideology. Formally,

$$g(p) = \begin{cases} (\lambda \cdot q(p) + (1 - \lambda) \cdot \mu(p)) \cdot \texttt{rec}(v) \cdot \texttt{ide}(v) & \text{if } \texttt{ide}(v) < 1/2 \\ (1 - \lambda \cdot q(p) - (1 - \lambda) \cdot \mu(p)) \cdot \texttt{rec}(v) \cdot (1 - \texttt{ide}(v)) & \text{if } \texttt{ide}(v) \geqslant 1/2. \end{cases}$$

[4] `http://en.wikipedia.org/wiki/NOMINATE_(scaling_method)`

Similarly, if the goal is to paint u as liberal, $g(p)$ can be defined as follows,

$$g(p) = \begin{cases} (\lambda \cdot q(p) + (1-\lambda) \cdot \mu(p)) \cdot \texttt{rec}(v) \cdot (1 - \texttt{ide}(v)) & \text{if } \texttt{ide}(v) < 1/2 \\ (1 - \lambda \cdot q(p) - (1-\lambda) \cdot \mu(p)) \cdot \texttt{rec}(v) \cdot \texttt{ide}(v) & \text{if } \texttt{ide}(v) \geqslant 1/2. \end{cases}$$

Among the three parameters open for perturbation, i.e., v, a, b, we would like to diversify the time interval of comparison, represented by a and b, for multiple representative points of the same target entity. We do not diversify the value of target entity v. Formally, let the diversification kernel $\mathcal{K}$ be a 3D Gaussian kernel with covariance matrix $\Sigma = \text{diag}(\sigma_e, \sigma_{\text{begin}}, \sigma_{\text{end}})$, where $\sigma_e = +\infty$.

4.5 Algorithms

In this section, we first present an iterative algorithm in Section 4.5.1 for solving the k-REPS problem. Multiple runs of the iterative algorithm with different seedings can provide a good guideline for the optimal value of the optimization objective $\mathcal{G}_{\mathcal{D}}$.

Motivated by efficiency, in Section 4.5.2, we propose a greedy approximation algorithm, which provides constant approximation of the optimal solution, due to the submodularity and monotonicity of the objective function $\mathcal{G}_{\mathcal{D}}$. The incremental nature of the greedy algorithm is helpful in determining the right value of k.

On top of the greedy algorithm, we present a number of techniques for further improving the efficiency and the result quality.

4.5.1 Iterative Algorithm

For the convenience of algorithm description, let $\mathcal{D}_S(s)$ denote, for $s \in S$, the subset of $\mathcal{D}$ represented by s among the set of representatives S, or formally

$$\mathcal{D}_S(s) = \{p \in \mathcal{D} \mid \phi_s(p) = \Phi_S(p)\}.$$

$\{\mathcal{D}_S(s)\}_{s\in S}$ is known as the maximization diagram of funtions $\{\phi_s\}_{s\in\S}$. The partition of the domain $\mathcal{D}$ is similar to Voronoi diagram (Voronoï, 1908), which is a minimization diagram of distances w.r.t. the representatives.

We present an iterative algorithm for solving the k-REPS problem, as an instance of the expectation-maximation (EM) algorithm (Dempster et al., 1977).

1. Initialize S to be an arbitrary k-subset of $\mathcal{D}$.
2. In each iteration:
    a. Compute the partition of $\mathcal{D}$ into clusters $\{\mathcal{D}_S(s)\}_{s\in S}$.
    b. For each cluster $\mathcal{D}_S(s)$, find the new representative point $s' = \arg\max_{s''\in\mathcal{D}} \sum_{p\in\mathcal{D}_S(s)} g(p)\cdot\phi_{s''}(p)$.
    c. Terminate if S is not updated.

Step 1 of the algorithm can be done in $O(k)$ time.

In Step 2a of the algorithm, computing the partition of $\mathcal{D}$ into clusters takes $O(k\cdot|\mathcal{D}|)$ time — computing $\phi_s(p)$ for all pairs of $s\in S$ and $p\in\mathcal{D}$, and picking the s maxmizing $\phi_s(p)$ for each p. In Step 2b, for each cluster, updating the best representative involves (i) enumerating the candidate s'', and (ii) evaluating $\sum_{p\in\mathcal{D}_S(s)} g(p)\cdot\phi_{s''}(p)$ for each s'' enumerated. The time complexity for each cluster is $O(|\mathcal{D}|\cdot|\mathcal{D}_S(s)|)$. Summing over all clusters, the overall time complexity of Step 2b is $O(|\mathcal{D}|^2)$. It is worth noting that when updating the representative of a cluster $\mathcal{D}_S(s)$, unlike the k-means clustering algorithm where the objective is to minimize the within-cluster sum of squared (Euclidean) distance, the new representative s' that maximizes the objective function $\mathcal{G}$ for points within the cluster may not necessarily be in the cluster.

The overall time complexity is dominated by the cost of updating the best representatives (Step 2b), and determined by number of iterations — $O(\#\text{iteration}\cdot|\mathcal{D}|^2)$.

Algorithm 6: Iterative-k-REPS$(g, \mathcal{D}, k)$.

```
S ← random k-subset of D;
while true do
    S_new ← ∅;
    foreach s ∈ S do
        D_S(s) = {p ∈ D | φ_s(p) ⩾ φ_s'(p), ∀s'};
        S_new ← S_new ∪ {arg max_{s'∈D} Σ_{p∈D_S(s)} g(p) · φ_s'(p)};
        if S = S_new then return S;
        S ← S_new;
```

Pseudo-code for the iterative algorithm is presented in Algorithm 6.

Remark (Choice of k)**.** If a cluster's optimal representative s' lies outside cluster, s' must have high quality expressed by $g(s')$ and be close to the cluster. This is a way suggesting too many representatives are selected.

Remark (Bad seeding)**.** The scenario described above — a cluster's best representative lies outside — may happen due to bad seeding of S, for example having two or more representatives very close to each other. Multiple runs of the iterative algorithm are needed, with different seeding of initial representative set S, in order not to wrongfully deeming the value of k to be too large due to bad seeding.

Instead of multiple trials of the iterative algorithm, the greedy algorithm to be presented next will help determine the right value of k as it grows the representative set in an incremental fashion.

Remark (Post-processing)**.** We adopt a gradient descent post-processing step to ensure that the solution set is locally optimal. For each $s \in S$, if there is a neighbor s' of s such that changing s unilaterally to s' (i.e., $S' = S \backslash \{s\} \cup \{s'\}$) would increase the value of $\mathcal{G}_\mathcal{D}$ (i.e., $\mathcal{G}_\mathcal{D}(S') > \mathcal{G}_\mathcal{D}(S)$), replace s with s' in S. Stop if there is no such neighbor for all representatives. Each update takes $O(k \cdot |\mathcal{D}|$ time. This process is guaranteed to terminate since each update increases the value of $\mathcal{G}_\mathcal{D}$, and the number of distinct k-subset of $\mathcal{D}$ is finite.

Algorithm 7: Greedy-k-REPS$(g, \mathcal{D}, k)$.

$S_0 \leftarrow \varnothing$;
for $i \leftarrow 1$ **to** k **do**
  $S_i \leftarrow S_{i-1} \cup \{\arg\max_{s \in \mathcal{D} \setminus S_{i-1}} \mathcal{G}_{\mathcal{D}}(S_{i-1} \cup \{s\})\}$;
return S_k;

4.5.2 Greedy Algorithm

One obvious drawback of the iterative algorithm is that, starting with a set of k random points, it may stuck at a locally maximal, but globally sub-optimal solution.

We present a greedy algorithm that grows the solution set incrementally without modifying existing solution points, and show that it generates an approximate solution with guarantee, due to the submodularity and monotonicity of the objective function $\mathcal{G}_{\mathcal{D}} : 2^{\mathcal{D}} \mapsto \mathbb{R}$.

The greedy algorithm (Algorithm 7) does exactly what the name suggests: starting out with an empty solution set S_0, in iteration i, find the point s to S that maximizes the objective function $\mathcal{G}_{\mathcal{D}}(S_i \cup \{s\})$. Choosing each greedy representative in a brute-force fashion takes $O(|\mathcal{D}|^2)$ time — $O(|\mathcal{D}|)$ for enumerating candidates s from $\mathcal{D}$, and $O(|\mathcal{D}|)$ for evaluating $\mathcal{G}_{\mathcal{D}}(S_i \cup \{s\})$ for each s.

The working set of representatives is append-only — no existing representative point is ever modified.

Submodularity and Monotonicity To establish the quality of solution by the greedy algorithm, we first show the submodularity (Schrijver, 2003) and monotonicity of the objective function $\mathcal{G}_{\mathcal{D}}$.

Definition 2 (*Submodularity*)**.** Let Ω be a set. A set function $f : 2^{\Omega} \mapsto \mathbb{R}$, where 2^{Ω} denotes the power set of Ω, is said to be *submodular* if for all $X \subseteq \Omega$ and $x_1, x_2 \in \Omega \setminus X$, we have

$$f(X \cup \{x_1\}) + f(X \cup \{x_2\}) \geqslant f(X \cup \{x_1, x_2\}) + f(X). \tag{4.20}$$

Lemma 1. *$\mathcal{G}_{\mathcal{D}}$ is a submodular set function, i.e., for all $S \subseteq \mathcal{D}$ and $s_1, s_2 \in \mathcal{D} \backslash S$, we have*

$$\mathcal{G}_{\mathcal{D}}(S \cup \{s_1\}) + \mathcal{G}_{\mathcal{D}}(S \cup \{s_2\}) \geqslant \mathcal{G}_{\mathcal{D}}(S \cup \{s_1, s_2\}) + \mathcal{G}_{\mathcal{D}}(S). \tag{4.21}$$

Proof. Consider two possibly intersecting subsets of $\mathcal{D}$, $P_i = \mathcal{D}_{S \cup \{s_i\}}(s_i)$, for $i = 1, 2$. First of all, it is clear that for all $p \in \mathcal{D} \backslash (P_1 \cup P_2)$, $\Phi_{S \cup \{s_i\}}(p) = \Phi_{S \cup \{s_1, s_2\}}(p) = \Phi_S(p)$, i.e., including either or both of s_1 and s_2 do not change the contribution of p to the objective function $\mathcal{G}$.

For $p \in P_1 \cup P_2$, we have

$$\max_{i=1,2}\{\Phi_{S \cup \{s_i\}}(p)\} = \Phi_{S \cup \{s_1, s_2\}}(p),$$

$$\min_{i=1,2}\{\Phi_{S \cup \{s_i\}}(p)\} \geqslant \Phi_S(p).$$

Adding up the equation and the inequality above, we have

$$\Phi_{S \cup \{s_1\}}(p) + \Phi_{S \cup \{s_2\}}(p) \geqslant \Phi_{S \cup \{s_1, s_2\}}(p) + \Phi_S(p).$$

Summing the equation above for all $p \in \mathcal{D}$, we get

$$\mathcal{G}_{\mathcal{D}}(S \cup \{s_1\}) + \mathcal{G}_{\mathcal{D}}(S \cup \{s_2\}) \geqslant \mathcal{G}_{\mathcal{D}}(S \cup \{s_1, s_2\}) + \mathcal{G}_{\mathcal{D}}(S).$$

Hence $\mathcal{G}_{\mathcal{D}}$ is submodular. □

Lemma 2. *$\mathcal{G}_{\mathcal{D}}$ is a monotone submodular function, i.e., for all $S_1, S_2 \subseteq \mathcal{D}$ such that $S_1 \subseteq S_2$, we have $\mathcal{G}_{\mathcal{D}}(S_1) \leqslant \mathcal{G}_{\mathcal{D}}(S_2)$*

The proof for Lemma 2 is trivial, thus omitted.

It was shown in Nemhauser et al. (1978) that the greedy algorithm provides a $(1 - 1/e)$-approximation for maximizing a monotone submodular set function with cardinality constraint.

Theorem 3. *Let S^* be the optimal solution to the k-REPS problem, and S^G be the solution by the greedy algorithm, we have*

$$\mathcal{G}_{\mathcal{D}}(S^G) \geqslant \left(1 - \frac{1}{e}\right) \cdot \mathcal{G}_{\mathcal{D}}(S^*)$$

A brute force implementation of the greedy algorithm enumerates s_i and for each enumerated s_i, evaluates $\mathcal{G}_{\mathcal{D}}(S_{i-1} \cup \{s_i\})$ in $O(|\mathcal{D}|)$ time, and picks the optimal s_i. The overall complexity is $O(k \cdot |\mathcal{D}|^2)$.

In contrast to the iterative algorithm (Algorithm 6), the number of iterations is bounded by k. As remarked in Section 4.5.1, if in iteration $i \leqslant k$, there exists $j < i$ such that $\phi_{s_j}(s_i) \geqslant \phi_{s_i}(s_i)$ or $\phi_{s_i}(s_j) \geqslant \phi_{s_j}(s_j)$, we consider the working set of representatives S_i under-diversified, and return S_{i-1} as the solution set instead of completing k iterations.

In the remaining of this section, we present a wide range of techniques for improving both efficiency and the quality of solution set (in terms of $\mathcal{GD}$) of the greedy algorithm, by utilizing the geometric properties of the objective function and the solution set.

Power Diagram Clustering Observe that in the greedy algorithm a representative selected into S will not change in future iterations. In the i-th iteration, when computing $\mathcal{G}_{\mathcal{D}}(S_{i-1} \cup \{s\})$ with an enumerated candidate s for s_i, we already knew $\mathcal{G}_{\mathcal{D}}(S_{i-1})$ from the previous iteration. To compute $\mathcal{G}_{\mathcal{D}}(S_{i-1} \cup \{s\})$ from $\mathcal{G}_{\mathcal{D}}(S_{i-1})$, we only need to consider points in $\mathcal{D}_{S_{i-1} \cup \{s\}}(s)$ as follows.

$$\mathcal{G}_{\mathcal{D}}(S_{i-1} \cup \{s\}) = \mathcal{G}_{\mathcal{D}}(S_{i-1}) + \sum_{p \in \mathcal{D}_{S_{i-1} \cup \{s\}}(s)} \phi_s(p) - \Phi_{S_{i-1}}(p). \tag{4.22}$$

We study the geometric structure of $\{\mathcal{D}_S(s)\}_{s \in S}$ with the Gaussian diversification kernel $\mathcal{K}$. Without loss of generality, let the covariance matrix $\Sigma = I$ for $\mathcal{K}$. Consider the bisector of the max-impact regions of two representatives s_1 and s_2.

$$
\begin{aligned}
& g(s_1) \cdot \mathcal{K}(p; s_1) = g(s_2) \cdot \mathcal{K}(p; s_2) \\
\Leftrightarrow & g(s_1) \cdot \exp(-\delta^2(p, s_1)/2) = g(s_2) \cdot \exp(-\delta^2(p, s_2)/2) \\
\Leftrightarrow & \exp\left(-\frac{1}{2}(\delta^2(p, s_2) - \delta^2(p, s_1))\right) = \frac{g(s_1)}{g(s_2)} \\
\Leftrightarrow & \delta^2(p, s_2) - \delta^2(p, s_1) = -2 \ln \frac{g(s_1)}{g(s_2)} \\
\Leftrightarrow & (p - s_2) \cdot (p - s_2) - (p - s_1) \cdot (p - s_1) = 2(s_1 - s_2)p - \|s_1\|_2^2 + \|s_2\|_2^2 = -2 \ln \frac{g(s_1)}{g(s_2)}.
\end{aligned}
$$

The results above suggests that the seperator of the max-impact regions of s_1 and s_2 in a d-dimensional parameter space $\mathcal{P}$ is a $(d-1)$-dimensional space. The max-impact region of representative s among a set of representatives S is the intersection of $|S| - 1$ halfspaces, thus a (possibly empty) d-dimensional convex polyhedron.

The max-impact regions of all representatives of S form a power diagram , a generalized form of Voronoi diagram (Aurenhammer and Klein, 2000). The formal definition of power diagram is presented as follows.

Definition 3. (*Power distance.*) Given a sphere of radius r and a point in a Euclidean space, with Euclidean distance d from the center of the sphere. The power distance from the point to the sphere is $d^2 - r^2$.

(*Power Diagram.*) Given a set of spheres in a Euclidean space, the power diagram is a partition of the space into polyhedral cells, where the cell for a given sphere C consists of all points for which the power distance to C is smaller than the power distance to all other spheres.

In the k-REPS problem, given a set of k representatives, S, consider the power diagram formed by k spheres, where the radius of the sphere centered at representative $s \in S$ is $\sqrt{2 \ln g(s)}$. Let $\mathrm{Vor}_S(s)$ denote the power diagram cell of s. The

power distance of $p \in \mathcal{D}$ to the sphere centered at s is $\delta^2(p,s) - \left(\sqrt{2\ln g(s)}\right)^2$. For each $s \in S$, we have $\mathcal{D}_S(s) = \mathrm{Vor}_S(s) \cap \mathcal{D}$. This supports the intuition behind the objective function that a representative s with higher individual quality, i.e., $g(s)$, should have higher representative power, reflected by larger sphere radius in the power diagram.

Back to Equation 4.22, the contribution of a candidate representative s to an existing representative set S can be written as

$$\mathcal{G}_{\mathcal{D}}(S \cup \{s\}) - \mathcal{G}_{\mathcal{D}}(S) = \sum_{p \in \mathcal{D}_{S\cup\{s\}}(s)} \phi_s(p) - \Phi_S(p). \tag{4.23}$$

If $S \cup \{s\}$ is a sufficiently diversified set of representatives, we have $s \in \mathcal{D}_{S\cup\{s\}}(s)$, i.e., s represents itself better than (or as well as) all representatives of S. As we have shown above, $\mathcal{D}_{S\cup\{s\}}(s) = \mathrm{Vor}_{S\cup\{s\}}(s) \cap \mathcal{D}$. Points of $\mathcal{D}_{S\cup\{s\}}(s)$ can be enumerated starting from s using the $\mathsf{GetP}_{\downarrow}$ subroutine described in Section 2.3.2.

Lazy Update We noticed that not all points are promising candidates for the solution set S. For example, in the same neighborhood of the problem domain $\mathcal{D}$, we would obviously prefer points with higher g-value. However, the basic greedy algorithm examines the possibility of every candidate being the next greedy solution point. Motivated by this observation, we introduce the lazy update technique, prioritizing the costly evalution of the contribution of a candidate solution point to the objective function $\mathcal{G}_{\mathcal{D}}$, resulting in possibly early decision of a candidate's inclusion in the greedy solution set, and reuses a lot of computation in future iterations.

Formally, let S_i denote the i-representative set by the greedy algorithm. For the boundary case, we define $S_0 = \varnothing$, and $\mathcal{G}_{\mathcal{D}}(\varnothing) = 0$. For a remaining candidate $p \in \mathcal{D}\backslash S_i$ after iteration i, let $\Delta_{p,i}$ denote the extra utility that would have been obtained by having p as the $(i+1)$-th representative in addition to S_i, i.e., $\Delta_{p,i} =$

$\mathcal{G}_{\mathcal{D}}(S_i \cup \{p\}) - \mathcal{G}_{\mathcal{D}}(S_i)$. The $(i+1)$-th representative to be chosen by the greedy algorithm is $s_{i+1} = \arg\max_{p \in \mathcal{D} \setminus S_i} \Delta_{p,i}$.

Observe that for all p, $j < i \Rightarrow \Delta_{p,j} \geqslant \Delta_{p,i}$. In other words, the earlier a candidate p is added to the incremental solution set, the more it can contribute to $\mathcal{G}_{\mathcal{D}}$. We use a priority queue Q to store triples $\langle p, j, \Delta_{p,j} \rangle$, denoting that if candidate p were to be added as the next solution point, at most $\Delta_{p,j}$ would be added to $\mathcal{G}_{\mathcal{D}}$, and the last time this extra utility was updated for p was after j-th iteration. At any time, for any p, at most one such triplet can exist in Q. The prioritization of elements of Q is based on the value of extra utility.

To find the $(i+1)$-th greedy representative, we repeatedly retrieve the top element $\langle p, j, \Delta_{p,j} \rangle$ from Q. If $j = i$, we know p must be the $(i+1)$-th greedy representative. Otherwise, we update $\Delta_{p,j}$ to $\Delta_{p,i}$, and re-insert triple $\langle p, i, \Delta_{p,i} \rangle$ into Q. To facilitate this update, we only need to consider $q \in \mathcal{D}_{S_i \cup \{p\}}(p) \bigcap \left(\bigcup_{l=i+1}^{j} \mathcal{D}_{S_j}(s_l) \right)$, which can be enumerated efficiently due to the power diagram structure of the max-impact regions of representatives discussed earlier.

For initialization, let $Q = \{\langle p, -1, g(p) \cdot \max_{q \in \mathcal{D}} g(q) \rangle \mid p \in \mathcal{D}\}$.

The lazy update technique will allow us to avoid trying lots of the candidates with low g-values, and candidates close to existing solutions, as new greedy solution point.

Non-diversification Subspace Algorithms 6 and 7 are presented in general terms, without considering the structure of the problem domain $\mathcal{D}$ or the diversification kernel $\mathcal{K}$. Now we show how to localize the greedy algorithm and improve its efficiency by utilizing the non-diversification subspace $\mathcal{P}_{\mathtt{ND}}$.

For a subset X of the parameter space (e.g. the problem domain $\mathcal{D}$, a set of representatives S), and $\breve{p} \in \mathcal{P}_{\mathtt{ND}}$, we define

$$X_{\check{p}} = \{p \in X \mid \pi_{\mathtt{ND}}(p) = \check{p}\}.$$

Due to the way $\mathcal{K}$ was defined in Section 4.2, i.e., $\mathcal{K}(p;s) > 0 \Rightarrow \pi_{\mathcal{P}_{\mathtt{ND}}}(p') = \pi_{\mathcal{P}_{\mathtt{ND}}}(s)$, we have

Lemma 4. $\mathcal{G}_{\mathcal{D}}\left(\bigcup_{\check{p}\in\mathcal{P}_{ND}} S_{\check{p}}\right) = \sum_{\check{p}\in\mathcal{P}_{ND}} \mathcal{G}_{\mathcal{D}_{\check{p}}}(S_{\check{p}})$

Corollary 5. *Let S^* be the optimal solution to the k-REPS problem on domain $\mathcal{D}$. For any $\check{p} \in \mathcal{P}_{ND}$, $S^*_{\check{p}}$ is the optimal solution to the $|S^*_{\check{p}}|$-REPS problem on sub-domain $\mathcal{D}_{\check{p}}$.*

Lemma 4 and Corollary 5 allows us to treat the k-REPS problem on the entire problem domain $\mathcal{D}$ as independent sub-problems of $k_{\check{p}}$-REPS on sub-domains $\mathcal{D}_{\check{p}}$, except that that optimal $k_{\check{p}}$ for each $\check{p}$ is not known apriori. However, this can be addressed by the incremental nature of the greedy algorithm. In each iteration of the greedy algorithm, we find the next greedy representative from each $\mathcal{D}_{\check{p}}$, and choose the best, in terms of the value of the objective function $\mathcal{G}_{\mathcal{D}}$. Local greedy solution points that are not chosen can be reused in the next iteration of the global greedy procedure.

Suppose there are m distinct projections of $\mathcal{D}$ on $\mathcal{P}_{\mathtt{ND}}$, leading to m partition of $\mathcal{D}$ of similar size. The time complexity is reduced to $O\left(\left(\frac{k}{m^2} + \frac{1}{m}\right) \cdot |\mathcal{D}|^2\right)$, where finding each locally optimal greedy solution point takes $O\left(\frac{|\mathcal{D}|^2}{m^2}\right)$ time. A total of $k+m$ such locally optimal greedy solution point is found, including one additional point for each of the m partition, on top of the k representatives asked by the k-REPS problem.

Size-limited Candidate Set We see that the complexity of the greedy algrorithm is heavily dependent on the number of candidates (linear without the lazy update

improvement). So far, we have not ruled out any point of $\mathcal{D}$ from being a candidate for S.

To significantly improve the efficiency of the greedy algorithm, we start with size-constrained candidate set, chosen randomly from $\mathcal{D}$. The probability of inclusion for each point is proportional to its g-value.

More precisely, to sample a candidate set $\mathcal{C} \subseteq \mathcal{D}$ of size $\eta \cdot |\mathcal{D}|$, where $\eta \in (0, 1]$, we first linearize elements of $\mathcal{D}$ as $p_1, \ldots, p_{|\mathcal{D}|}$, in an arbitrary order. Define the cdf $G_i = G_{i-1} + g(p_i)$ for $i = 1, \ldots, |\mathcal{D}|$, with $G_0 = 0$. To select a candidate, we generate x from the continuous uniform distribution unif $\left(0, \sum_{p \in \mathcal{D}} g(p)\right)$, and include p_i in $\mathcal{C}$ where $i = \min\{j > 0 \mid G_j \geqslant x\}$, if p_i does not exist in $\mathcal{C}$ yet. We repeat this step until the size of $\mathcal{C}$ reaches $\eta \cdot |\mathcal{D}|$.

This reduction of the candidate set may limit the quality of the solution set. To address this problem, we introduce the following post-processing step.

Post-processing using Iterative Algorithm On top of all above techniques to speed up the greedy algorithm, we add one extra iteration of the iterative algorithm as a post-processing step, seeded with the greedy solution, in order to improve the quality of solution, in terms of $\mathcal{G}_{\mathcal{D}}$. All points in $\mathcal{D}$ are candidates again in this step.

The Combined Algorithm Now we present an improved greedy algorithm, called Greedy+ (Algorithm 8), incorporating all technique described above.

As an adaptation of notations from Algorithm 7, to account for partitions of $\mathcal{D}$ by their projections on the non-diversification $\mathcal{P}_{\mathsf{ND}}$, let $\Delta_{p,i} = \mathcal{G}_{\mathcal{D}_{\check{p}}}(S_{\check{p},i} \cup \{p\}) - \mathcal{G}_{\mathcal{D}_{\check{p}}}(S_{\check{p},i})$, where $\check{p}$ is the projection of p on $\mathcal{P}_{\mathsf{ND}}$

A smaller candidate set $\mathcal{C}$ is used instead of the full domain $\mathcal{D}$ (Line 13). To account for the non-diversification subspace $\mathcal{P}_{\mathsf{ND}}$, we run independent instances of the greedy algorithm for each distinct projection $\check{p}$ on $\mathcal{P}_{\mathsf{ND}}$ (Lines 14 – 17).

Lines 20 through 28 describes the lazy update process. A priority queue $Q_{\check{p}}$ is used to realize the lazy update for each possible projection $\check{p}$ on $\mathcal{P}_{\mathsf{ND}}$. A merge queue Q_M is used to select the maximum element for the next lazy update operation across different values of $\check{p}$. In the LazyUpdate subroutine for updating $\Delta_{p,j}$ to $\Delta_{p,i}$, enumeration of points on Lines 5 and 9) are performed efficiently based on the power diagram structure of representatives' max-impact regions.

Lines 30 through 33 describes the post-processing iteration.

4.6 Experiments

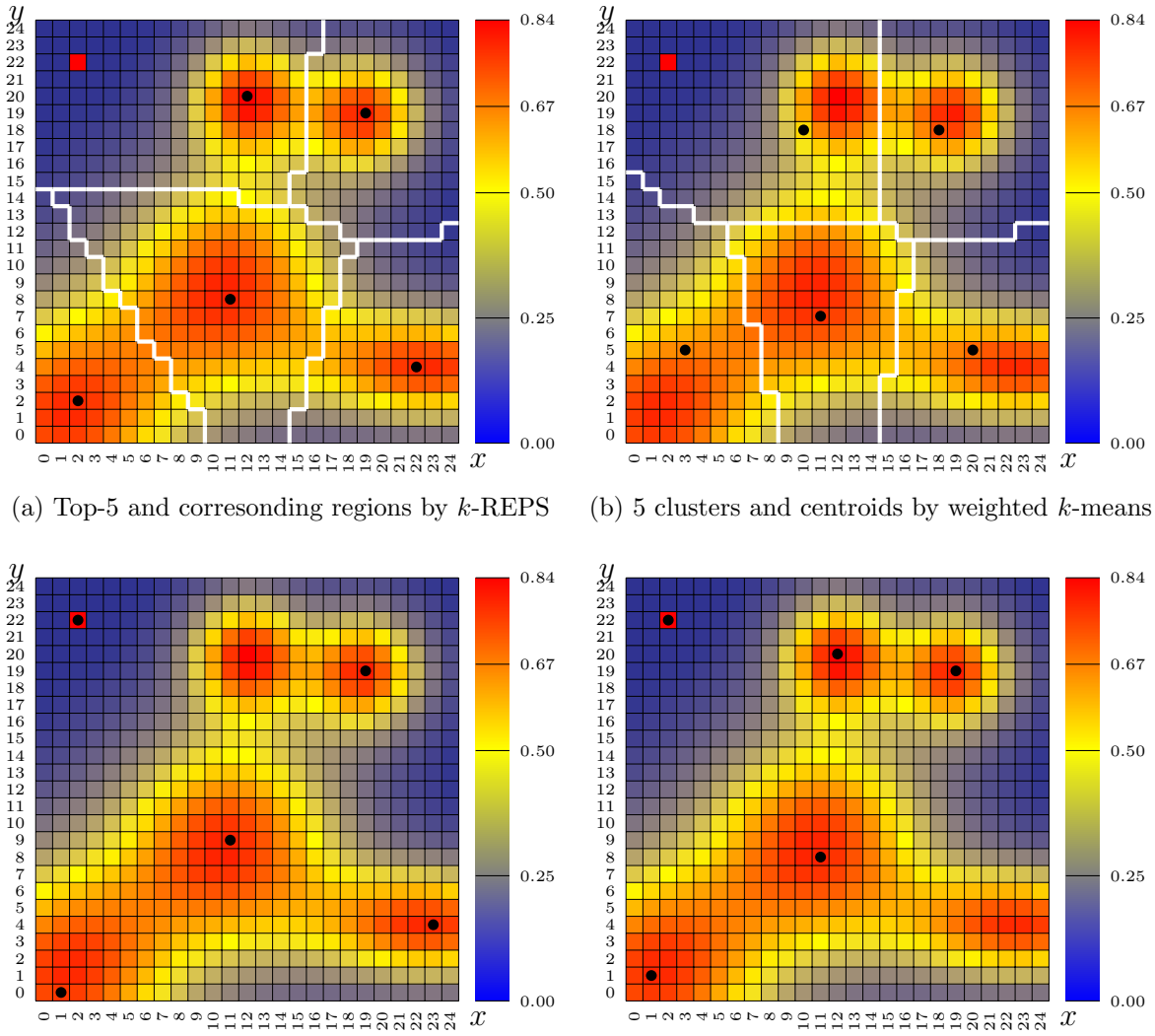
In this Section, we first compare k-REPS with two methods of diversified top-k and the weighted k-means clustering algorithm on a synthetic surface. Then we present the results of k-REPS on ***UNEMPY***, the US yearly unemployment rate data, for 65 years from 1948 to 2012 ($n = 65$), derived by taking the average monthly unemployment rate (***UNEMP***, Section 3.4) by calendar years.

To evaluate the efficiency and effectiveness of the proposed algorithms, we use the US Congressional voting data, ***VOTE*** as described in Section 3.4, which has a larges parameter space compared to ***UNEMPY***.

4.6.1 Proof-of-concept Experiments

Comparison with Related Work Our high level objectives are closely to the missions of two class of problems, namely top-k queries with diversification and clustering. We compare the quality of results with existing work on a synthetic surface (Figure 4.2), as a mixture of five Gaussian kernels with similar peak values, and a single spike point $(3, 22)$ in a low-value region.

k-REPS vs. k-means. To see the difference between the k-REPS problem and clustering problem, we first compare with the classical (weighted) k-means clustering



(a) Top-5 and corresonding regions by k-REPS

(b) 5 clusters and centroids by weighted k-means

(c) Top-5 by MMR

(d) Top-5 by graph based approach

FIGURE 4.2: Top-k representative points by different methods on synthetic surface

algorithm (MacQueen, 1967; Lloyd, 1982). Figure 4.2a shows the optimal five representative points according to Definition 1 given $k = 5$. The green curves partition the surface into regions that are best represented by each of the representatives. Figure 4.2b shows the five clusters by weighted k-means clustering, with the surface value as each point's weight. The (weighted) centroid of each cluster is rounded to the nearest integral point and shown using a black dot. The clustering by k-REPS and by k-means are not too different from each other. However, the k-means defi-

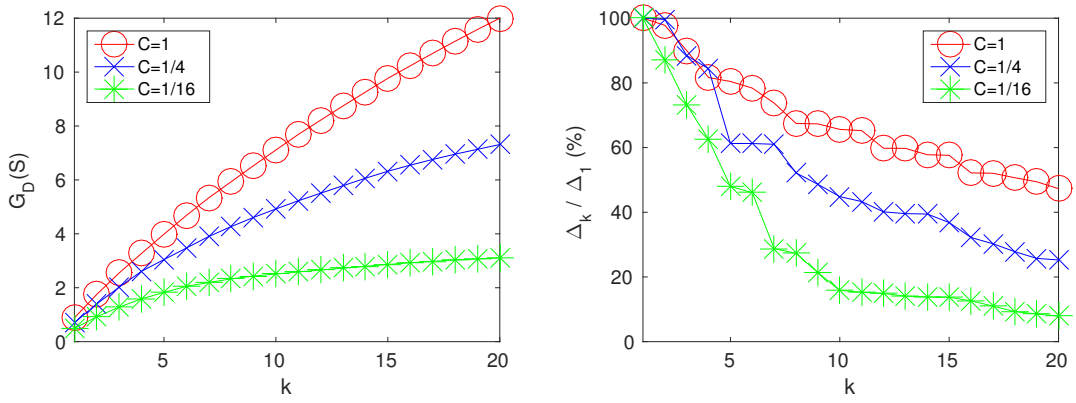
nition which aims to minimize the *within-cluster sum of squares* results in possibly low-weight centroids, i.e., low-quality representatives in our context. This is likely to happen when the high-value region of a cluster deviates from the unweighted center of the cluster, e.g. the top-left, bottom left, and bottom right clusters represented by points $(10, 18)$, $(3, 5)$, and $(20, 5)$, respetively in Figure 4.2b.

k-REPS vs. diversified top-k. We also compare with diversified top-k results on the same synthetic surface, shown in Figure 4.2c and 4.2d. Figure 4.2c shows the result of the popular MMR approach Carbonell and Goldstein (1998); Catallo et al. (2013), which iteratively returns the next representative point p from $\mathcal{D} \backslash S$ that optimizes the following function, as a linear combination of utility [5] and diversity.

$$\lambda \cdot g(p) - (1 - \lambda) \cdot \max_{s \in S} \mathsf{sim}(s, p) \tag{4.24}$$

In Equation 4.24 above, $\lambda \in (0, 1]$ dictates the trade-off between utility and diversity. At one extreme, when $\lambda = 1$, all focus is on utility, and points will be returned by MMR in non-ascending order of their g values. On the other hand, when $\lambda \to 0$, only the highest g value point will be returned because any other point as a second representative would yield negative value in Equation 4.24. For the purpose of this experiment, we choose the λ that gives us exactly five solution points, i.e., any candidate for 6th representative would lead to negative value of Equation 4.24. We see in Figure 4.2c that the solution points picked up by MMR tends to be pushed towards the boundary of the candidate solution space (e.g. points $(1, 0)$ and $(23, 4)$), because diversity is expressed explicitly as part of the optimization objective. Also, MMR does not consider the relationship between solution points and non-solution points, and as a result, the high-value point $(2, 22)$ in a low-value region is returned by MMR.

[5] Note that the utility of a candidate point has been adapted from its similarity/relevance to a query point, to its g value.

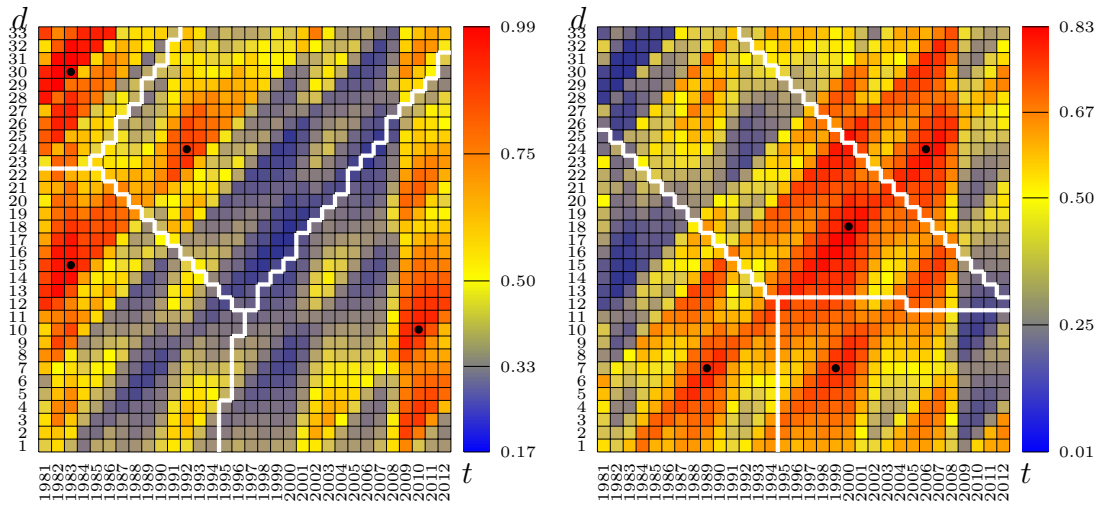


(a) $\mathcal{G}(S_k)$ as k changes. (b) Δ_k/Δ_1 as k changes.

FIGURE 4.3: Using the greedy algorithm to guide the selection of the number of representatives k

Another examplary work in diversified top-k takes the similarity-graph approach Qin et al. (2012) – two candidate points are connected by an edge, thus mutually exclusive in the solution set, if they are close to each other, dictated by a distance threshold δ. The goal is to find a independent set of size at most k that maximizes the sum of utilities, i.e., g-values. The result by this similarity graph based approach on the same synthetic surface is shown in Figure 4.2d. Similar to the result of MMR, point $(2, 22)$ is returned because the relationship between solution points and non-solution points is not considered.

It it worth noting that the results of k-REPS, MMR, and the graph based method are the same in extreme cases, but they do not reduce each other when the trade-off of utility and diversity is in-between. The maximum diversity is achieved (i) in k-REPS when $C \to +\infty$ in $\mathcal{K}$; (ii) in MMR when $\lambda \to 0$; (iii) in the graph-based formulation when distance threshold $\delta = 0$. When diversity is maximized, all three methods return only the highest utility point. At the other extreme, when $C = 0$ in $\mathcal{K}$ for k-REPS, $\lambda = 1$ for MMR, and $\delta \to \infty$ for the graph-based approach, all three methods would return the k points with highest utility.



(a) Aiming for high response. (b) Aiming for low response.

FIGURE 4.4: 4-REPS for finding WAC claims on unemployment rate data

Case Study: Finding WAC Claims on UNEMPY Now we turn to a concrete example, of finding WAC claims on ***UNEMPY***. Per discussion in Section 3.2.1, the parameter space of WAC claims on ***UNEMPY*** consists of triples (w, t, d) satisfying that $w \in [1, n-1]$, $t \in [w+1+1948, n+1948]$[6] and $d \in [1, t-w]$. For the purpose of visualization, we fix the window size $w = 1$, and consider t, d open for perturbation. More precisely, let $\mathcal{D} = \{(w, t, d) \mid w = 1 \wedge t \in [1981, 2012] \wedge d \in [1, 33]\}$.[7] Let $g(p) = \tilde{q}(p)$ for $p \in \mathcal{D}$, where q was defined in Eqn. 3.1, the normalized using the logistic function as described in Section 4.3. Our goal is to look for high-response representatives.

For the diversification kernel $\mathcal{K}$, let $\mathcal{K}(p; s) = \exp\left\{-\frac{C}{2}(p-s)^T I(p-s)\right\}$. Note that the choice of $\Sigma = I$ here is different from that in Section 3.2.1, because in lead finding, there is no need for capturing the context of a given claim.

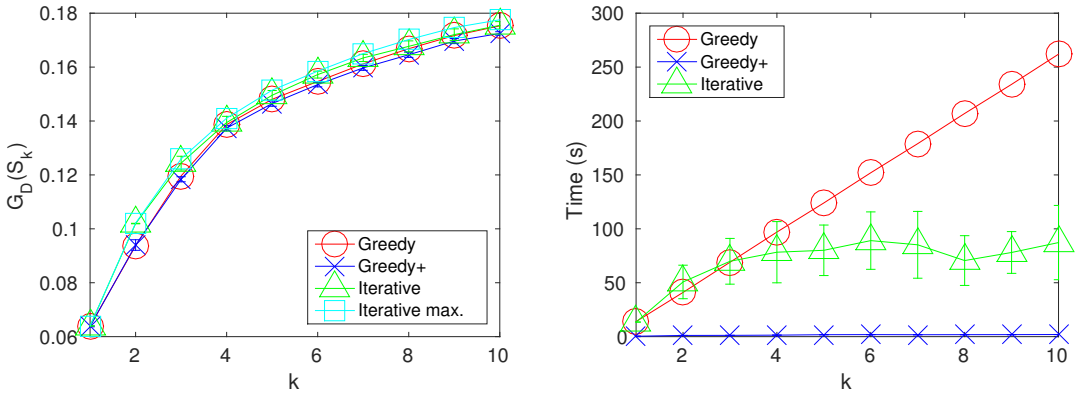
[6] Note that ***UNEMPY*** starts at the year of 1948. We add an offset of 1948 to the range of t, instead of using the general range definition on an arbitrary time series, for better interpretability.

[7] A 2D slice of the 3D parameter space for WACclaims with fixed value of w forms a triangle. An analogous example was shown in Figure 2.1 for WAC on the New York City's adoption data. This definition of $\mathcal{D}$ corresponds to a lower right rectangular region of the triangle.

We first show the how to use the greedy algorithm to guide the selection of the value of k, and the effect of C on diversity of results. In Figure 4.3, we plot the increase of value of the objective function $\mathcal{G}$ as the number of representatives k increases, for three different values of C. Let S_k be the greedy solution to k-REPS. In Figure 4.3a, we plot the value of the greedy solution sets $\mathcal{G}_{\mathcal{D}}(S_k)$. In Figure 4.3b, we plot the value of each additional greedy representative, as a percentage of the value of the first representative, i.e., Δ_k/Δ_1, where $\Delta_k = \mathcal{G}_{\mathcal{D}}(S_k) - \mathcal{G}_{\mathcal{D}}(S_{k-1})$. We see that the contribution of each additional greedy representative decreases as k increases, and the rate of decrease increases as C decreases. As discussed in Section 4.2, smaller value of C corresponds to higher diversity. Hence, fewer representatives are needed to the cover the same domain. For a fixed value of C, a good value of k can be chosen such that the next greedy representative gives significantly smaller contribution. For example, for $C = 1/4$, $k = 4$ can be a good choice, as the contribution Δ_5 of the fifth greedy representative is only 60% of Δ_1, as opposed to $> 80\%$ by the fourth greedy representative.

In Figure 4.4a, we plot the greedy 4-REPS and their corresponding max-impact regions on the QRS, normalized on domain $\mathcal{D}$. It is clear that all four representatives is have high values by themselves, and different high-value regions of the surface. Here are the four WAC claims translated from the four representatives.

- The unemployment rate in 1983 increased by as much as 169.8% compared to that of 1968.
- The unemployment rate in 1983 increased by as much as 228.2% compared to that of 1953.
- The unemployment rate in 1992 increased by as much as 110.5% compared to that of 1968.



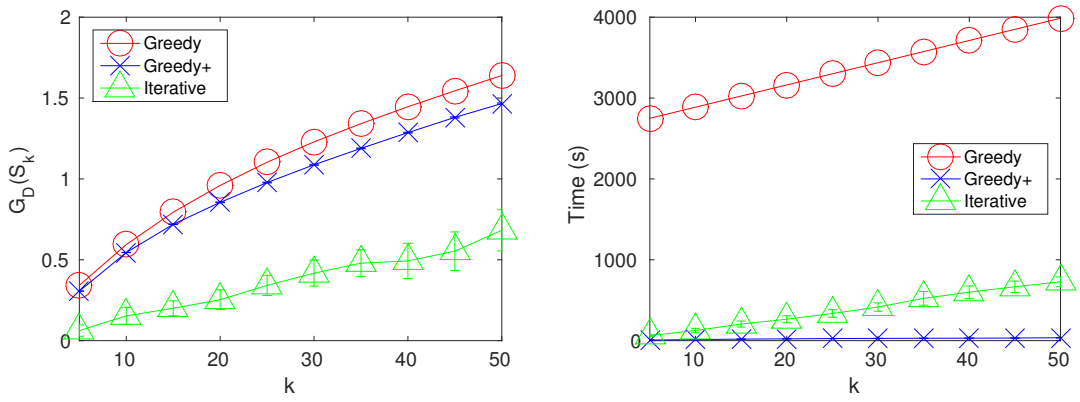
(a) Effectiveness (b) Efficiency

FIGURE 4.5: Performance of k-REPS algorithms (Marshall vs. Pelosi)

- The unemployment rate in 2010 increased by as much as 142.6% compared to that of 2000.

Conversely, we can reverse the normalized response, i.e., $g(p) = 1 - \tilde{q}(p)$, to look for low-response claims. The results of 4-REPS are shown in Figure 4.4b. The corresponding WAC claims boasting decrease in unemployment are as follows.

- The unemployment rate in 1989 decreased by as much as 45.8% compared to that of 1982.
- The unemployment rate in 1999 decreased by as much as 43.7% compared to that of 1992.
- The unemployment rate in 2000 decreased by as much as 59.1% compared to that of 1982.
- The unemployment rate in 2006 decreased by as much as 52.5% compared to that of 1982.



(a) Effectiveness (b) Efficiency

FIGURE 4.6: Performance of k-REPS algorithms (Marshall vs. all)

4.6.2 Algorithm Effectiveness and Efficiency

We compare the efficiency and the effectiveness of the algorithms proposed in Section 4.5, using the US Congressional voting data (***VOTE***). The modeling of k-REPS problem for finding TSS claims on ***VOTE*** was discussed in Section 4.4. We focus on votes from 2001 to 2014. There are 153 distinct entities/legislators with at least one appearance on Sunday news shows (the recognizability measure as discussed in Section 4.4). Restricting the time of comparison to whole months, we have a total of 154 months with at least one vote.

For a TSS claim parameterized as (u, v, a, b) with fixed value of u, the size of the domain $\mathcal{D}$ is roughly $154 \cdot (154 \text{ choose } 2) \approx 1.8 \times 10^6$. With the non-diversification dimension v, $\mathcal{D}$ can be partitioned into 154 2D sub-domains, each of size roughly 1.2×10^4.

We compare the three algorithms proposed in Section 4.5, namely, the Iterative Algorithm (Algorithm 6), the Greedy Algorithm (Algorithm 7), and the Greedy+ Algorithm (Algorithm 8).

We first find k representative claims that paint James Marshall as liberal by claiming a high vote correlation with the notable Democrat Nancy Pelosi, for $k =$

$1, 2, \ldots, 10$. This sub-domain of $\mathcal{D}$ is 2-dimensional, without non-diversification subspace $\mathcal{P}_{\mathtt{ND}}$. Results are shown in Figure 4.5. Results of the Iterative Algorithm are based on 20 random k-subsets of $\mathcal{D}$ as seeds. Results of the Greedy+ are based on 20 independent runs with random candidate sets with 5% of $\mathcal{D}$, sampled as described in Section 4.5.2. In Figure 4.5a, we see that the performance of the three algorithms (average performance for Greedy+ and Iterative) are comparable to each other, in terms of $\mathcal{G}_{\mathcal{D}}(S_k)$. The best case result of Iterative is slightly better than the other two algorithms. However, we see in Figure 4.5b that Greedy+ runs much faster than both Greedy and Iterative. The running time of Greedy increases linearly in k, while the running time of Iterative depends on the number of iterations until convergence, thus hard to predict. If time is not a constraint, repeated runs of Iterative may improve the quality of the resulting representative set.

In Figure 4.6, we show the result of k-REPS on the full 3-dimensional domain $\mathcal{D}$, with the target entity being the non-diversification dimension, as described in Section 4.4. k ranges from 5 to 50. The goal is the same — to paint Marshall as liberal, by claiming low vote correlation with notable conservatives, or high vote correlation with other notable liberals. In Figure 4.6a, we see that Iterative is outperformed by Greedy and Greedy+. Recall that a non-diversification subspace $\mathcal{P}_{\mathtt{ND}}$ effectively divides the k-REPS problem into m independent sub-problems for the m distinct projections of $\mathcal{D}$ on $\mathcal{P}_{\mathtt{ND}}$ (Corollary 5). However, for the global optimal k-REPS solution S^*, the number of representatives that belong to a projection $\check{p} \in \mathcal{P}_{\mathtt{ND}}$ is not known apriori. With the Iterative algorithm, a bad seeding — wrong distribution of number of representatives for each possible projection — will yield sub-optimal solution. This issue does not exist for Greedy or Greedy+ due to their incremental behavior. In terms of efficiency, Greedy+, again, is faster than the other two (Figure 4.6b). The Greedy algorithm has a big overhead cost upon finding the first representative because it needs to find m greedy representatives, one for each of the

m distinct projections on $\mathcal{P}_{\mathsf{ND}}$.

Overall, we see in both Figure 4.5 and Figure 4.6 that Greedy+ is more efficient than Iterative and Greedy, due to a smaller candidate set. While Iterative is not efficient by itself, having it as a post-processing step for Greedy+ guards against poor result quality from the smaller candidate set, and brings the quality of result of Greedy+ close to that of Greedy.

4.7 Related Work

The lead-finding problem studied in this chapter is closely related to a large body of work on diversification of results for recommender systems and top-k queries.

Top-k diversification As discussed in Section 4.6.1, Carbonell and Goldstein (1998) proposed the *maximal marginal relevance* method that iteratively retrieves the next "optimal" item with respect to the existing solution set in a greedy fashion. Optimality was defined as the trade-off, in the form of a linear combination, between an item's relevance to the query item (utility), and its relevance to existing solution items (diversity). The highest utility item (most relevant to query item) will always be selected as the first item in the solution set. Catallo et al. (2013) later built on MMR and studied the diversified top-k problem over bounded regions.

Qin et al. (2012) took a graph-based approach towards diversifying top-k results. Two items are mutually exclusive in the solution set if they close similar to each other. The goal is to maximize the total utility of no more than k items.

While these two methods focus on the quality of solution points only, we consider the relationship between the solution points and the non-solution ones, by maximizing the impact of the solution set on the entire problem domain.

Weighted Clustering We have also compared the k-REPS problem with the popular weighted k-means clustering algorithm (MacQueen, 1967; Lloyd, 1982). While the weighted k-means algorithm focuses on clustering instead of choosing high-weight centroids, the centroids may have low weight when the weighted and unweighted centroids of a cluster are far from each other.

Ackerman et al. (2012) studied the classification of weighted clustering algorithms. k-means belongs to the class of *weight sensitive* weighted clustering algorithm. It can be shown that the partition of max-impact regions of representatives by the k-REPS problem is also weight sensitive.

Diversification in recommender systems The utility-diversity trade-off also frequently appears in recommender systems. Zhou et al. (2010) studied the trade-off through (a linear combination of) two different probability re-distribution methods using collaborative filtering. Diversification is achieved via prefering "weak-ties".

Hasan et al. (2014) studied the problem of adaptive diversification in search result nagivation. Instead of presenting all solution items at once with a fixed level of diversity, they modeled displaying search result as an interactive process, and aimed for adaptive diversity based on user response.

Spatial diversification Item relevance in recommender systems can be regarded as item distance in a metric space, or parameter space distance in our context. Jain et al. (2004); Haritsa (2009) studied the problem of k-Nearest Diverse Neighbor (kNDN) from a geometric perspective.

Sensor placement Guestrin et al. (2005) studied the problem of sensor placement with budget. The goal was to cover a space with a limited number of sensors to increase the confidence of temperature prediction based on readings from place sen-

sors. Spatial diversification was achieved as a result of maximizing global information gain, i.e., minimize uncertainty. Interestingly, it was shown that this optimization objective is equivalent to maximizing *mutual information* between solution set and non-solution set, which is analogous to our motivation of non-solution point coverage.

Image Segmentation k-REPS problem on a 2-dimensional domain is related to *image segmentation*. The goal of image segmentation is to identify boundaries of objects in images. A wide range of methods have been proposed for image segmentation, such as clustering methods (Pappas, 1992), compression-based methods (Mobahi et al., 2011; Rao et al., 2010), histogram-based methods (Ohlander et al., 1978), edge detection and integration methods (Kimmel, 2003; Kimmel and Bruckstein, 2003), variational methods (Mumford and Shah, 1989; Chan and Vese, 2001), etc.

In image segmentation, a high-value singular point will be identified as a region or segment by itself. For this reason, image segmentation algorithms cannot be readily applied to in lead-finding, to find areas of high-quality leads while avoiding singular points.

Submodular set function Submodular set functions (Schrijver, 2003) have found applications in many domains, including but not limited to *document summarization* (Lin and Bilmes, 2011), *image collection summarization* (Tschiatschek et al., 2014), *sensor placement* (Guestrin et al., 2005), *feature selection* and *active learning* (Krause and Guestrin, 2008).

Maximization of submodular functions is usually NP-hard. Nemhauser et al. (1978) have shown that greedy algorithms provide $(1 - 1/e)$-approximation to maximization problem of a monotone submodular set function with cardinality constraint. More generally, a $\frac{1}{2}$-approximation algorithm exists for submodular function maximization problem without monotonicity or cardinality constraint (Buchbinder et al.,

2012).

Algorithm 8: Greedy$^+$-k-REPS$(g, \mathcal{D}, k)$.

def LazyUpdate$(\check{p}, p, j, i, \Delta_{p,j})$ **begin**
  if $\forall s \in S_{\check{p},i}$, $\phi_p(p) \geqslant \phi_s(p)$ **and** $\phi_s(s) \geqslant \phi_p(s)$ **then**
    if $j = -1$ **then**
      $\Delta_{p,i} \leftarrow 0$;
      foreach $q \in (\mathcal{D}_{\check{p}})_{S_{\check{p},i} \cup \{p\}}(p)$ **do**
        $\Delta_{p,i} \leftarrow \Delta_{p,i} + g(q) \cdot \phi_p(q)$;
    else
      $\Delta_{p,i} \leftarrow \Delta_{p,j}$;
      foreach $q \in (\mathcal{D}_{\check{p}})_{S_{\check{p},j} \cup \{p\}}(p) \bigcap \left(\bigcup_{l=j+1}^{i} (\mathcal{D}_{\check{p}})_{S_{\check{p},i}}(s_l) \right)$ **do**
        $\Delta_{p,i} \leftarrow \Delta_{p,i} - g(q) \cdot \left(\min\{\Phi_{S_{\check{p},i}}(q), \phi_p(q)\} - \Phi_{S_{\check{p},j}}(q) \right)$;
    $Q_{\check{p}}$.insert$(\langle p, i, \Delta_{p,i} \rangle)$;
  return;
$\mathcal{C} \leftarrow$ random subset of $\mathcal{D}$ of size $\eta \cdot |\mathcal{D}|$;
foreach $\check{p} \in \mathcal{P}_{ND}$ **do**
  $\mathcal{C}_{\check{p}} \leftarrow \{q \in \mathcal{C} \mid \pi_{\mathcal{P}_{\mathsf{ND}}}(q) = \check{p}\}$;
  $Q_{\check{p}} \leftarrow \{\langle p, -1, g(p) \cdot \max_{q \in \mathcal{D}_{\check{p}}} g(q) \rangle \mid \forall p \in \mathcal{D}_{\check{p}}\}$;
  $k_{\check{p}} \leftarrow 0$; $S_{\check{p},0} \leftarrow \varnothing$;
$Q_M \leftarrow \{\langle \check{p}, Q_{\check{p}}.\mathsf{removeMax}() \rangle\}_{\check{p} \in \mathcal{P}_{\mathsf{ND}}}$;
for $i \leftarrow 1$ **to** k **do**
  while true do
    if Q_M.*empty*() **then break**;
    $\langle \check{p}, \langle p, j, \Delta_{p,j} \rangle \rangle \leftarrow Q_M$.removeMax();
    if $j = k_{\check{p}}$ **then**
      $S_{\check{p},k_{\check{p}}+1} \leftarrow S_{\check{p},k_{\check{p}}} \cup \{p\}$;
    else
      LazyUpdate$(\check{p}, p, j, k_{\check{p}}, \Delta_{p,j})$;
    if $\neg Q_{\check{p}}$.*empty*() **then** Q_M.insert$(\langle \check{p}, Q_{\check{p}}.\mathsf{removeMax}() \rangle)$;
    if $j = k_{\check{p}}$ **then** $k_{\check{p}} \leftarrow k_{\check{p}} + 1$; **break** ;
$S \leftarrow \varnothing$;
foreach $\check{p} \in \mathcal{P}_{ND}$ **do**
  $S_{\check{p}} \leftarrow S_{\check{p},k_{\check{p}}}$;
  foreach $s \in S$ **do**
    $S \leftarrow S \cup \{\arg\max_{s' \in \mathcal{D}_{\check{p}}} \sum_{p \in (\mathcal{D}_{\check{p}})_{S_{\check{p}}}(s)} g(p) \cdot \phi_{s'}(p)\}$;
return S;

5

Lead-Finding : One-of-the-few Claims

5.1 Introduction

One important goal in computational journalism is *(semi-)automatic lead identification* from data, i.e., finding interesting information nuggets from raw data that lead to further investigation and/or news stories around them. In Chapter 4, we have studied the k-REPS problem that serves as a general tool for finding diversified high-quality leads.

In this chapter, we shift the focus to finding high-quality leads driven by quality measures, in particular, *uniqueness*, as defined in Section 2.2.2 (Eqn. 2.3). We consider a popular form of claims exemplified by the following:

- *There is no player in NBA history with more points, more rebounds, and more assists than Oscar Robertson in one's career.*
- *Rick Perry is one of the only three candidates in the 2012 US federal election cycle to have received at least $600k from "lawyers & lobbyists" (an interest group that is usually pro-Democrat) and $400k from "energy & natural resources" (usually pro-Republican).*

These two claims share a common structure: both are about an object being one of the few that "stand out" when compared according to a set of numeric attributes. More precisely, given a set of objects $\mathcal{O}$, each with a set $\mathcal{A}$ of numeric attributes, a *one-of-the-few* claim has the following form:

> ***Object*** o ***is dominated by fewer than*** k ***objects in a non-empty subset*** $\mathcal{B} \subseteq \mathcal{A}$ ***of attributes.***

Here, we say o' dominates o in $\mathcal{B}$ if o' is no worse than o for all attributes in $\mathcal{B}$, and o' is strictly better than o for at least one attribute in $\mathcal{B}$. Journalists are interested in two tasks: 1) finding all "interesting" one-of-the-few claims from a given dataset; 2) ranking objects based on what one-of-the-few claims can be made about them. The first task allows journalists to identify claims that can be used in stories or serve as leads for further investigation. The second task provides an object-centric view that allows journalists to prioritize their investigation of particular objects (especially when there are many interesting one-of-the-few claims).

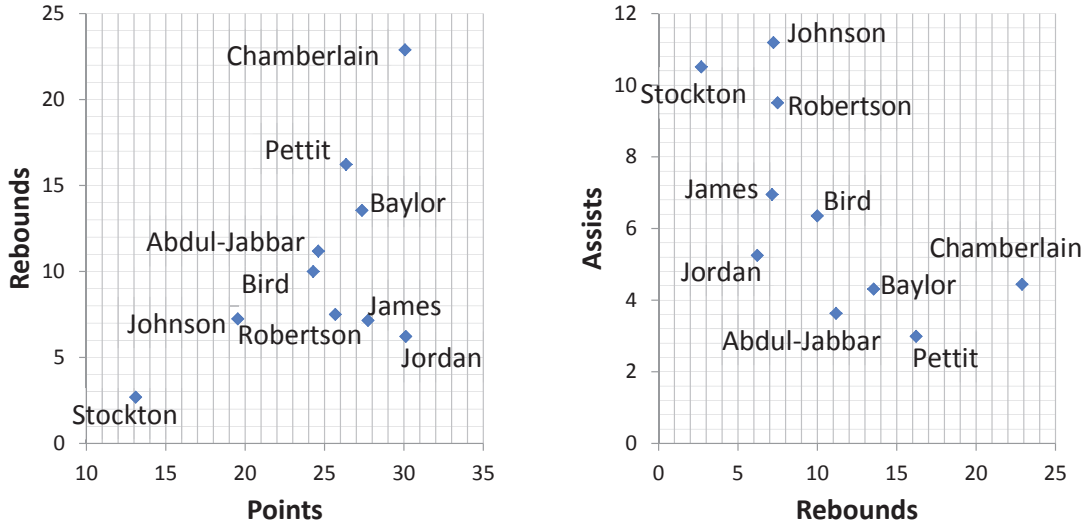
Task 1: Finding Claims One-of-the-few claims are closely related to the concept of *k-skyband* for multi-dimensional data (Papadias et al., 2005). Intuitively, the k-skyband of a set $\mathcal{O}$ of objects in subspace $\mathcal{B}$ is the subset of objects each dominated by fewer than k other objects in $\mathcal{B}$. (The better-known notion of *skyline* is a special case of k-skyband where $k = 1$.) From the k-skyband in $\mathcal{B}$, a one-of-the-few claim can be generated for each object in the skyband straightforwardly. While various algorithms exist for computing a k-skyband given k and $\mathcal{B}$, they do not address the key challenge of ensuring "interestingness" of the claims they find in a way that is easy for users to control and interpret. Although the parameter k can be tuned, it is a poor indicator of interestingness, as illustrated by the following example.

Example 4. Consider the set of nearly 4000 players in NBA history, with stats such as career total points, rebounds, and assists. Being one of the top 50 leading scorers—i.e., in the 50-skyband for the single-attribute subspace {points}—is quite an impressive feat. However, if we expand the subspace to {points, rebounds}, 142 players now fit the bill—i.e., each is dominated by fewer than 50 others in {points, rebounds}. If we further include assists in the subspace, 324 (almost 9% of all) players will be in the 50-skyband.

Example 4 clearly illustrates why we cannot use a universal k to ensure interestingness of one-of-the-few claims for different subspaces: the size of the skyband tends to increase rapidly as dimensionality goes up. For example, the expected number of 1-skyband (skyline) objects is $O(\ln^{d-1}|\mathcal{O}|/(d-1)!)$ for a d-dimensional subspace (Bentley et al., 1978), assuming no correlation between attributes. With a fixed k, too many claims can be in high-dimensional subspaces, making them less interesting. Clearly, k needs to be adjusted when $|\mathcal{B}|$ changes. Furthermore, the appropriate setting for k cannot be expressed simply as a function of dimensionality, because data characteristics—for example, correlation among attributes—also matter, as illustrated below.

Example 5. We plot a subset of NBA players, with attributes points, rebounds, and assists per game, as points in subspaces {points, rebounds} (Figure 5.1a) and {rebounds, assists} (Figure 5.1b). It is easy to see that the skybands in Figure 5.1a tend to be smaller than those in Figure 5.1b, because of the positive correlation between points and rebounds and the negative correlation between rebounds and assists. For example, the 3-skyband in {points, rebounds} contains 5 players (Jordan, Chamberlain, James, Baylor, and Pettit), which translate into 5 one-of-the-3 claims; on the other hand, the 3-skyband in {rebounds, assists} contains 9 players (all except Jordan). Hence, no single choice of k is appropriate for these two subspaces of the same dimensionality.

Example 5 above clearly illustrates that we cannot hope to define interestingness,



(a) Positive: *points* vs. *rebounds* (b) Negative: *rebounds* vs. *assists*

FIGURE 5.1: Correlation in NBA player stats.

which is data-dependent, by a function of k and $|\mathcal{B}|$ alone. Asking the user to pick the right k manually for each and every subspace is also infeasible. Our quest is to find an effective way of ensuring claim interestingness such that: 1) users are not required to tune lots of parameters; 2) the results are easy to understand and explain in layman's terms. Both properties are critical for computational journalism, where journalists may be non-technical and the results need to be explained to the general public in stories.

Task 2: Ranking Objects Even with an appropriate definition of "interestingness," many objects may be the subject of at least one interesting one-of-the-few claim in some subspace. The task of ranking objects allows users to prioritize their effort in investigating objects. From our experience analyzing real data and preliminary user studies, different data domains and user preferences call for some degree of customization in ranking, as illustrated below.

Example 6. Both John Stockton and Larry Bird are inductees into the Naismith Memorial Basketball Hall of Fame, but they have very different playing styles. Stockton has the second highest assists per game in NBA history, but is not very impressive in points or rebounds. Bird ranks 17th in points, 60th rebounds, and 44th in assists. Stockton and Bird exemplify what we call "specialized" and "well-rounded" objects, respectively. How to rank specialized objects relative to well-rounded ones often depends on the context in which the ranking will be used, or may simply be a matter of personal opinion.

A popular method for ranking objects according to multiple attributes is *Kemeny optimal rank aggregation* (Dwork et al., 2001), or *Kemeny* for short. It produces a "consensus" ranking that minimizes the number of pairwise disagreements (in the relative ordering of two objects) with respect to the rankings under individual attributes. Kemeny tends to downgrade objects that rank extremely high in very few attributes but considerably low in other attributes. For Example 6 above, Bird, who is well-rounded, would be ranked as the 9th by Kemeny, while Stockton, who is specialized, would be as low as the 139th, which may not be acceptable to some.

While Kemeny leaves no option for customization, another popular method, *weight-sum ranking*, exposes too many knobs. With weighted-sum, a user specifies a preference vector, whose components represent weights assigned to individual attributes; objects are then ranked according to their projection onto this vector (i.e., weighted combination of their attribute values). For a d-dimensional dataset, the user needs to properly specify d weights; even if we learn these weights automatically, training examples must be provided by the user. Such requirements may overwhelm journalists with little time or technical expertise. Our goal is to devise a ranking scheme with as few knobs as possible, which would allow customization without overwhelming users.

Main Contributions First, we propose a simple but effective definition for the interestingness of a claim based on its "uniqueness." For a one-of-the-few claim (with a particular k in a particular subspace $\mathcal{B}$) to be interesting, we require that this claim (with the same k and $\mathcal{B}$) cannot be made for more than τ objects. Unlike k, τ is a user-defined threshold that applies universally to all subspaces, significantly reducing the burden on the user to define interestingness. For each subspace, our definition automatically adapts k in a data-dependent way, and naturally excludes those high-dimensional subspaces where no claims are unique enough. Furthermore, τ is also easy to understand for non-technical users.

Based on this definition, we introduce the problem of finding all interesting one-of-the-few claims across all non-empty subspaces, given the uniqueness threshold τ. The fact that our k is data-dependent and not fixed raises unique challenges not addressed by previous work on computing skylines and skybands. We devise efficient algorithms that avoid redundant computation. In particular, we are able to improve the worst-case complexity of finding all interesting claims in a subspace from $O(|\mathcal{O}|^2)$ to $O(\tau|\mathcal{O}|)$, which is attractive because τ in practice is small for claims to be unique.

Building on the definition of interestingness, we propose a novel scheme for scoring and ranking objects based on the aggregated interestingness of claims involving them. One key insight distinguishing our scheme from others such as Kemeny and weighted-sum is that we in effect aggregate ranks across all non-empty subspaces as opposed to just individual attributes. Our scheme supports tuning by a single parameter α, which captures user preference between specialized and well-rounded objects, and overcomes the inflexibility of Kemeny without resorting to an overwhelming number of knobs like weighted-sum. Extending the algorithms for finding all interesting claims, we show how to compute top-ranked objects efficiently given α. We experimentally demonstrate, on real datasets, that our scheme is able to produce rankings comparable to Kemeny and weighted-sum while offering more effective

customization.

5.2 Finding One-of-the-Few Claims

Preliminaries Consider a set $\mathcal{O}$ of n objects, each with d numeric attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$. A *subspace* is a subset of the attributes and the set of all subspaces of $\mathcal{A}$ form a lattice. We say that subspace $\mathcal{B}_1$ is an *ancestor* (*descendant*) of subspace $\mathcal{B}_2$ if $\mathcal{B}_1 \subseteq \mathcal{B}_2$ (resp. $\mathcal{B}_1 \supseteq \mathcal{B}_2$). We say $\mathcal{B}_1$ is a *parent* (*child*) of $\mathcal{B}_2$ if $\mathcal{B}_1 \subseteq \mathcal{B}_2$ (resp. $\mathcal{B}_1 \supseteq \mathcal{B}_2$) and their cardinalities differ by one.

We say o_1 *dominates* o_2 in subspace $\mathcal{B}$, denoted $o_1 \succ_{\mathcal{B}} o_2$, if i) $\forall A \in \mathcal{B}, o_1.A \geqslant o_2.A$, and ii) $\exists A \in \mathcal{B}, o_1.A > o_2.A$. Clearly, dominance is transitive: if $o_1 \succ_{\mathcal{B}} o_2$ and $o_2 \succ_{\mathcal{B}} o_3$, then $o_1 \succ_{\mathcal{B}} o_3$.

Definition 4 (Dominating Subset, k-Skyband, Skyline, Tier)**.**

- The *dominating subset* of $o \in \mathcal{O}$ in subspace $\mathcal{B} \subseteq \mathcal{A}$, denoted $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o)$, is the subset of objects that dominate o in $\mathcal{B}$; i.e., $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o) = \{o' \in \mathcal{O} \mid o' \succ_{\mathcal{B}} o\}$. Let $\delta_{\mathcal{B}}(\mathcal{O}, o) = |\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o)|$ denote the size of the dominating subset.

- The *k-skyband* ($k \geqslant 1$) of $\mathcal{O}$ in subspace $\mathcal{B}$, denoted $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$, is the subset of objects in $\mathcal{O}$ that are each dominated by fewer than k other objects in $\mathcal{O}$; i.e. $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}) = \{o \in \mathcal{O} \mid \delta_{\mathcal{B}}(\mathcal{O}, o) < k\}$.

- The *skyline* of $\mathcal{O}$ in subspace $\mathcal{B}$ is $\mathfrak{S}^1_{\mathcal{B}}(\mathcal{O})$, i.e., the 1-skyband.

- The *i-th tier* ($i \geqslant 1$) of $\mathcal{O}$ in subspace $\mathcal{B}$ is the subset of objects in $\mathcal{O}$ that are each dominated by exactly $i-1$ other objects in $\mathcal{O}$; i.e. $\{o \in \mathcal{O} \mid \delta_{\mathcal{B}}(\mathcal{O}, o) = i-1\}$.

Clearly, by definition, the k-skyband $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$ is the disjoint union of all i-th tiers with $i \leqslant k$, and the difference between the k-skyband and the $(k-1)$-skyband is the k-th tier. [1]

[1] Note that tiers differ from the well-known concept of *maximal layers*, where the next maximal layer is defined as the skyline of the set of objects outside all previous layers.

To illustrate, consider the set $\mathcal{O}$ of 10 NBA players and the subspace $\mathcal{B} = \{\mathsf{rebounds}, \mathsf{assists}\}$ shown in Figure 5.1b. $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = \{\text{Johnson}\}$, so $\delta_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = 1$. $\mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) = \{\text{Johnson}, \text{Robertson}, \text{Bird}, \text{Chamberlain}\}$ is the skyline (or 1-skyband), which is also the 1st tier. $\mathfrak{S}^2_{\mathcal{B}}(\mathcal{O}) = \mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) \cup \{\text{Stockton}, \text{Baylor}, \text{Pettit}\}$ is the 2-skyband, where Stockton, Baylor, and Pettit are in the 2nd tier and each dominated by exactly one object in $\mathcal{O}$. $\mathfrak{S}^3_{\mathcal{B}}(\mathcal{O})$, the 3-skyband, additionally includes the 3rd tier $\{\text{James}, \text{Abdul-Jabbar}\}$, leaving only Jordan out, as mentioned in Example 5.

Problem Statement As motivated in Section 5.1, while each object in the k-skyband translates into a one-of-the-few claim, we measure the interestingness of this claim by the number of objects for which similar claims can be made, i.e., the size of the k-skyband. Instead of struggling with setting k, which depends on the subspace and object distribution, a user should be able to specify a single threshold τ that caps the number of similar claims. Therefore, we introduce the concept of *top-τ skyband* below. While the concept is closely related to k-skyband, a crucial difference is that a top-τ skyband is defined by its size, while a k-skyband, defined by its number of tiers, can be arbitrarily large.

Definition 5 (Top-τ Skyband)**.** Given $\tau \geqslant 1$, the *top-τ skyband* of a set of objects $\mathcal{O}$ in subspace $\mathcal{B}$ is the largest skyband whose size does not exceed τ. In other words, it is the $\hat{k}$-skyband where $\hat{k} = \max\{k \mid \tau \geqslant |\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})|\}$.

The fact that an object o belongs to the top-τ skyband with the k-th tier as its last non-empty tier—or alternatively, the k-skyband with size no more than τ—translates into the following statement:

> *Object o is dominated by fewer than k objects in $\mathcal{B}$, and this claim cannot be made for more than τ objects.*

Intuitively, τ measures the uniqueness of the claim made by the first part of the statement above. For example, in Figure 5.1a, suppose we set $\tau = 3$. The top-3 skyband is the 1-skyband $\{\text{Chamberlain}, \text{Jordan}\}$. The 2-skyband would be too big, because it additionally contains Pettit, Baylor, and James and has size $5 > 3 = \tau$. Note that the 3rd tier is empty—no player is dominated by exactly two others in this example—so the 3-skyband (recall Example 5) is the same as the 2-skyband.

The problem of finding all interesting one-of-the-few claims can now be formulated as follows:

Definition 6 (Finding Top-τ Skybands in All Subspaces)**.** Given the set $\mathcal{O}$ of objects and a user-specified threshold τ, find, for every non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, the top-$\tau$ skyband of $\mathcal{O}$ in $\mathcal{B}$.

For each subspace, the membership of an object in the top-τ skyband corresponds a one-of-the-few claim that cannot be made for more than τ objects. This definition leads naturally to some desired features. In a single-attribute subspace, the top-τ skyband contains essentially the top k objects ranked by this attribute (but with better handling of ties). As the subspace dimensionality goes up, k decreases in an automatic, data-dependent manner until the k-skyband contains no more than τ objects. Thus, with this problem formulation, users do not need to pick k manually for different subspaces. To illustrate, consider again Example 5. Suppose the user sets $\tau = 8$. For subspace $\{\mathsf{points}, \mathsf{rebounds}\}$ (Figure 5.1a), the top-8 skyband would be the 5-skyband. In contrast, for $\{\mathsf{rebounds}, \mathsf{assists}\}$ (Figure 5.1b), the top-8 skyband would be the 2-skyband. Finally, in very high-dimensional subspaces where so many objects are on the skyline that no claims are interesting, our problem formulation correctly leads to an empty top-τ skyband.

Parameter Space Representation In the QRS-based framework, given a non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, the parameter space can be defined as the finite set of objects, i.e. $\mathcal{P} = \mathcal{O}$. Let the response function q be defined as

$$q(o) = \delta_{\mathcal{B}}(\mathcal{O}, o)\,, \quad \forall o \in \mathcal{P}. \tag{5.1}$$

Let the relative result strength function be defined simply as

$$\mathsf{SR}(r; r_0) = r_0 - r. \tag{5.2}$$

It is not hard to see that the top-τ skyband of $\mathcal{O}$ in $\mathcal{B}$ is equivalent to the set of one-of-the-few claims represented by $\langle q, o, q(o)\rangle$ whose *uniqueness* (defined in Eqn. 2.3) are no less than $1 - \frac{\tau}{n}$.

Overview of Solutions The rest of this section describes our algorithms for finding all interesting one-of-the-few claims. At a high level, we 1) traverse the lattice of subspaces in some manner, and 2) compute the top-τ skyband for each subspace we visit. Techniques for improving efficiency exist both across subspaces and within each subspace. For finding the top-τ skyband in a given subspace $\mathcal{B}$, we propose two algorithms in Section 5.2 and 8. We then show in Section 5.2.1 how to explore the lattice of subspaces using these algorithms as subroutines.

Note that computing the top-τ skyband for a single-attribute subspace $\{A\}$ simply involves finding the top τ objects sorted by A; algorithms in Sections 5.2 and 8 are needed only when $|\mathcal{B}| > 1$.

Also note that it is possible to devise solutions by adapting existing techniques from literature. We present one such solution here, which we call Baseline. For lattice traversal, Baseline adopts the strategy of *bottom-up skyline (BUS)* of Pei et al. (2006), who study the problem of computing the skyline in every subspace.

Algorithm 9: Progressive$(\mathcal{O}, \mathcal{B}, \tau)$

Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold τ
Result: the top-τ skyband of $\mathcal{O}$ in $\mathcal{B}$

```
Φ ← ∅; k ← 0;
while true do
    S ← 𝔖¹_ℬ(𝒪\Φ);
    k′ ← min{δ_ℬ(Φ, o) + 1 | o ∈ S};
    ΔΦ ← {o ∈ S | δ_ℬ(Φ, o) + 1 = k′};
    if |Φ ∪ ΔΦ| > τ then break;
    Φ ← Φ ∪ ΔΦ; k ← k′;
return Φ;
```

BUS can be extended to compute k-skyband if k is given. In each subspace, Baseline starts with $k = 1$, and computes the k-skyband and increments k, iteratively, until the k-skyband contains at least τ objects. Obviously, this iterative process of finding the right k leads to a lot of redundant computation. Our new algorithms avoid such redundant computation, and we experimentally validate their advantages over Baseline in Section 5.4.

Progressive Top-τ Skyband Algorithm

As it turns out, all objects in the $(k+1)$-th tier must lie on the skyline of the set of remaining objects after those in the k-skyband are taken out. This simple but useful observation has probably been made in other contexts too; we state it as a lemma here for completeness.

Lemma 6. $\mathfrak{S}_{\mathcal{B}}^{k+1}(\mathcal{O}) \setminus \mathfrak{S}_{\mathcal{B}}^{k}(\mathcal{O}) \subseteq \mathfrak{S}_{\mathcal{B}}^{1}\big(\mathcal{O} \setminus \mathfrak{S}_{\mathcal{B}}^{k}(\mathcal{O})\big)$.

Proof. Let $\overline{\mathfrak{S}}$ denote $\mathcal{O}\setminus\mathfrak{S}_{\mathcal{B}}^{k}(\mathcal{O})$. Consider any o in the $(k+1)$-th tier. Clearly $o \in \overline{\mathfrak{S}}$ because $\delta_{\mathcal{B}}(\mathcal{O}, o) = k$. If $o \notin \mathfrak{S}_{\mathcal{B}}^{1}\big(\overline{\mathfrak{S}}\big)$, then there exist $o' \in \overline{\mathfrak{S}}$ such that $o' \succ_{\mathcal{B}} o$. By transitivity of dominance, every object that dominates o' also dominates o, so $\delta_{\mathcal{B}}(\mathcal{O}, o) > \delta_{\mathcal{B}}(\mathcal{O}, o') \geqslant k$, a contradiction. □

Lemma 6 suggests the following strategy, which we call Progressive (Algorithm 9),

for computing the top-τ skyband for a given subspace. Given τ, Progressive computes the answer set Φ tier by tier, starting from the first. By Lemma 6, to obtain the next non-empty tier, we first compute (Line 3) the skyline S for the set of remaining objects (those outside the current Φ). Objects in the next non-empty tier ($\Delta\Phi$ on Line 5) are those in S whose dominating subsets in $\mathcal{O}$ are the smallest in size. We add this tier to Φ as long as the resulting answer set has no more than τ objects. It is easy to see that at the end of each iteration, the invariant that $\Phi = \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$ holds. Progressive terminates if the addition of the next non-empty tier causes the size of the answer set to exceed τ.

Note that on Lines 4 and 5 we compute the size of the dominating subset for $o \in S$ in Φ instead of $\mathcal{O}$. This optimization is correct because, being on the skyline of $\mathcal{O}\backslash\Phi$, o is not dominated by any object in $\mathcal{O}\backslash\Phi$, so $\delta_{\mathcal{B}}(\Phi, o) = \delta_{\mathcal{B}}(\mathcal{O}, o)$.

Further Optimizations For clarity of presentation, Algorithm 9 leaves out some details, which we describe below. Suppose that in the previous iteration, the skyline computed was $\check{S}$, and $\Delta\check{\Phi} \subseteq \check{S}$ was added to the answer set. For the current iteration, it is easy to see that $\check{S}\backslash\Delta\check{\Phi} \subseteq S$; i.e., all remaining objects in $\check{S}$ will appear again in the skyline S to be computed. This observation leads to two optimizations. **1)** We can speed up successive skyline computations on Line 3 across iterations. Specifically, we adapt the *SUBSKY* algorithm of Tao et al. (2006) (although in general any skyline algorithm can be used). The original SUBSKY uses an index whose size is linear in the $|\mathcal{O}|$ to help compute the skyline of $\mathcal{O}$ in any subspace. During execution, SUBSKY always maintains the skyline for the subset of objects that it has examined. In our adaption of SUBSKY, we remove from (a copy of) the SUBSKY index any object added to Φ, and we "seed" each invocation of SUBSKY with $\check{S}\backslash\Delta\check{\Phi}$ instead of starting it with an empty skyline. **2)** We can reduce the number of dominance tests involved in determining $\delta_{\mathcal{B}}(\Phi, o)$ for $o \in S$ on Lines 4 and 5. For any $o \in \check{S}\backslash\Delta\check{\Phi} \subseteq S$,

we have already computed $\delta_{\mathcal{B}}(\mathcal{O}, o)$ in the previous iteration, so there is no need to recompute it.

Complexity Following convention (Tao et al., 2006), we measure the performance of our algorithms by the number of dominance tests. Progressive performs two types of such tests: 1) those involved in computing the skyline of the remaining objects, and 2) those involved in computing the sizes of the dominating subsets in $\mathcal{O}$ for objects on that skyline. The number of type-1 tests depends on the skyline algorithm; Tao et al. (2006) describes various techniques to reduce it for SUBSKY. However, in the worst case, $|\Phi|$ is almost τ in the final iteration, while $|S|$ can be roughly $|\mathcal{O}| - \tau$ (i.e., the next non-empty tier contains nearly all remaining objects). In this case, the total number of dominance tests would be $\Theta(|\mathcal{O}|^2)$.

One-Pass Top-τ Skyband Algorithm

Progressive has poor worst-case complexity: it computes the entire next tier in order to decide whether to add that tier to the answer, but that tier can be much larger than τ. In this section, we show how to tame the complexity in terms of τ with another algorithm OnePass. This algorithm works by considering object one by one in a particular order while maintaining an answer set Φ. Dominance tests are only performed between the object being considered and those in the current Φ. The processing order is chosen in a "safe" way, as defined below, which allows OnePass to cap $|\Phi|$ at τ at all times, thereby bounding the number of dominance tests to $\tau|\mathcal{O}|$.

Definition 7 (Safe Order)**.** A order for a set $\mathcal{O}$ of objects is *safe* (for OnePass) if o' precedes o whenever $o' \succ_{\mathcal{B}} o$.

Algorithm 10 describes OnePass. The details of implementing the safe order will be given later in this section. Here, we first explain the algorithm and establish its correctness, the crux of which is captured by the lemma below.

Lemma 7. *The following invariants are true at the end of each iteration of* OnePass*'s main loop, where $\mathcal{O}^\star$ denotes the set of all objects processed so far.*

(I-1) For all $o' \in \Phi$, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$.

(I-2) $\Phi = \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}^\star)$.

(I-3) $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})| > \tau$ (if $|\mathcal{O}| > \tau$).

Proof. We prove the lemma by induction. All invariants obviously hold before the beginning of the first iteration. Suppose all invariants hold before the start of an iteration processing o. We show that they also hold when the current iteration ends.

For clarity, let $\check{\Phi}$ and $\check{k}$ denote the values of Φ and k, respectively, at the start of the current iteration; let $\check{\mathcal{O}}^\star = \mathcal{O}^\star \backslash \{o\}$ denote the set of objects processed before the current iteration.

(I1): At the end of the iteration, for any $o' \in \Phi$ that was also in $\check{\Phi}$, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$ stays true as OnePass does not update $c[o']$. It remains to be shown that if o is added to Φ, then $c[o] = \delta_{\mathcal{B}}(\mathcal{O}, o)$. Note that $c[o] = \delta_{\mathcal{B}}\big(\check{\Phi}, o\big) < \check{k}$. It suffice to show that all objects in $\mathcal{O}$ dominating o are in $\check{\Phi}$. Suppose that is not the case. Let o' denote the first object in safe order such that $o' \succ_{\mathcal{B}} o$ and $o' \notin \check{\Phi}$. Because of the safe processing order, we know $o' \in \check{\mathcal{O}}^\star$, and $\delta_{\mathcal{B}}\big(\check{\mathcal{O}}^\star \backslash \check{\Phi}, o'\big) = 0$. Meanwhile, $\delta_{\mathcal{B}}\big(\check{\Phi}, o'\big) \leqslant \delta_{\mathcal{B}}\big(\check{\Phi}, o\big) < \check{k}$. Therefore, $\delta_{\mathcal{B}}\big(\check{\mathcal{O}}^\star, o'\big) = \delta_{\mathcal{B}}\big(\check{\mathcal{O}}^\star \backslash \check{\Phi}, o'\big) + \delta_{\mathcal{B}}\big(\check{\Phi}, o'\big) < \check{k}$, implying that $o' \in \mathfrak{S}^{\check{k}}_{\mathcal{B}}\big(\check{\mathcal{O}}^\star\big)$. However, by the inductive hypothesis, $\mathfrak{S}^{\check{k}}_{\mathcal{B}}\big(\check{\mathcal{O}}^\star\big) = \check{\Phi}$, contradicting the assumption that $o' \notin \check{\Phi}$.

(I2): First, for any object $o' \in \check{\mathcal{O}}^\star$, o' cannot be dominated by o because of the safe processing order. Therefore, $\delta_{\mathcal{B}}\big(\check{\mathcal{O}}^\star, o'\big) = \delta_{\mathcal{B}}(\mathcal{O}^\star, o')$, which implies $o' \in \mathfrak{S}^j_{\mathcal{B}}\big(\check{\mathcal{O}}^\star\big) \Leftrightarrow o' \in \mathfrak{S}^j_{\mathcal{B}}(\mathcal{O}^\star)$ for any j. To complete the proof we need to consider the possible membership of o in Φ. There are two cases. Case 1: o is not added to Φ because $\delta_{\mathcal{B}}\big(\check{\Phi}, o\big) \geqslant \check{k} = k$. In this case, $\delta_{\mathcal{B}}(\mathcal{O}^\star, o) \geqslant \delta_{\mathcal{B}}\big(\check{\Phi}, o\big) \geqslant k$, which means $o \notin \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}^\star)$.

It is easy to verify that (I2) holds. Case 2: o is added to Φ. As shown in the proof for (I1), $c[o] = \delta_{\mathcal{B}}(\mathcal{O}, o) < \check{k}$, and all objects in $\mathcal{O}$ dominating o are in $\check{\Phi} \subseteq \mathcal{O}^\star$. Therefore, $o \in \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$. It is easy to verify that $\Phi = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$ right after Line 8. If Lines 9 –11 are not executed, $k = \check{k}$ and (I2) obviously holds. Otherwise, consider Lines 9– 11. For any $o' \in \Phi$ right before Line 11, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$ as shown by the proof for (I1). Because of the safe processing order, no object in $\mathcal{O}\backslash\mathcal{O}^\star$ dominates o', so $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}^\star, o')$. Therefore, Lines 9–11 in effect remove the last non-empty tier of $\Phi = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$ and update k accordingly, so (I2) also holds.

(I3): If k is not updated in the current iteration, (I3) obviously remains valid. Suppose k is updated. As shown in the proof for (I2) above, $\check{\Phi} \cup \{o\} = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$, and Line 11 simply removes the last non-empty tier of this $\check{k}$-skyband, which is the $(k+1)$-th tier where $k+1 \leqslant \check{k}$. Hence, $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star)| = |\mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)| = |\check{\Phi} \cup \{o\}| > \tau$ (by Line 9). Because of the safe processing order, no object in $\mathcal{O}^\star$ is dominated by any in $\mathcal{O}\backslash\mathcal{O}^\star$, so for all $o' \in \mathcal{O}^\star$, $\delta_{\mathcal{B}}(\mathcal{O}^\star, o') = \delta_{\mathcal{B}}(\mathcal{O}, o')$; therefore, $\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star) \subseteq \mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})$, so $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})| \geqslant |\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star)| > \tau$. □

Consider the next object o in safe order. OnePass first checks whether o is dominated by at least k objects in Φ (Lines 3–6). If yes, o is ignored because it would be in the $(k+1)$-th or later tier (by (I-1)) and therefore cannot be in the top-τ skyband of $\mathcal{O}$ (by (I-3)). We stop counting as soon as $c[o]$ reaches k. Heuristically, we check o against "better" objects in Φ (i.e., those with smaller dominating sets) first, in hope of reaching k sooner with fewer dominance tests.

If $c[o] < k$, OnePass adds o to Φ (Lines 8) and remember the count $c[o]$. If doing so makes $|\Phi|$ exceed τ we remove the last tier from Φ and update k accordingly (Lines 9– 11), to preserve (I-2) and (I-3). If k drops to 0 (Line 12), that means even the skyline has size bigger than τ (by (I-3)), so we can terminate (and return $\varnothing$) without processing the remaining objects.

Algorithm 10: OnePass($\mathcal{O}, \mathcal{B}, \tau$)

Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold τ
Result: the top-τ skyband of $\mathcal{O}$ in $\mathcal{B}$

```
1  Φ ← ∅; k ← τ;
2  foreach o ∈ O in safe order (Definition 7) do
3      c[o] ← 0 ; // count the number of objects in Φ dominating o
4      foreach o' ∈ Φ do // heuristically in ascending order of c[o']
5          if o' ≻_B o then c[o] ← c[o] + 1;
6          if c[o] ⩾ k then break;
           // no need to count further
7      if c[o] < k then
8          Φ ← Φ ∪ {o};
9          if |Φ| > τ then // Φ is too big; remove the last tier
10             k ← max{c[o'] | o' ∈ Φ};
11             Φ ← Φ\{o' ∈ Φ | c[o'] = k};
12             if k = 0 then break;
13 return Φ;
```

After all objects in $\mathcal{O}$ are processed, Φ is the top-τ skyband of $\mathcal{O}$ in $\mathcal{B}$, by (I-2) and (I-3).

Producing a Safe Order The simplest approach for OnePass to produce a safe processing order is to sort $\mathcal{O}$ by $\mathcal{B}$, which would take $O(|\mathcal{O}| \log |\mathcal{O}|)$ time. To avoid the cost of a full sort, we precompute, for each of the d attributes in $\mathcal{A}$, a version of $\mathcal{O}$ sorted by that attribute. Given a subspace $\mathcal{B}$, OnePass picks the attribute $A \in \mathcal{B}$ with the largest domain (a heuristic also adopted in Beyer and Ramakrishnan (1999); Pei et al. (2006)), and use the version of $\mathcal{O}$ sorted by A. During processing, if OnePass finds that a group of objects tie in A, OnePass further sorts this group by the full $\mathcal{B}$. As a further optimization, before sorting this group, OnePass first performs Lines 3–6 on each object o in the group, and removes those with $c[o] \geqslant k$; the filtered group is then sorted and further processed. In the worst case, OnePass still needs to pay $O(|\mathcal{O}| \log |\mathcal{O}|)$ for sorting. In practice, however, we have found our heuristics very effective in eliminating sorting, because most ties tend to occur later in processing;

by that time, most objects will be eliminated by Lines 3–6. Furthermore, the cost of filtering is low because τ and k are typically small.

Complexity Because OnePass caps $|\Phi|$ at τ at all times, the number of dominance tests is bounded by $\tau|\mathcal{O}|$. As explained above, sorting adds $O(|\mathcal{O}|\log|\mathcal{O}|)$ in the worst case, though in practice the extra cost is rarely incurred. Furthermore, In high-dimensional subspaces, k decreases rapidly as OnePass examines more objects. It is likely that an object will be discarded after a few dominance tests. OnePass also detects the case when the size of the skyline would exceed τ, and is able to terminate without examining the whole $\mathcal{O}$. Thus, OnePass is expected to run much faster in practice than the bound suggests, particularly for high-dimensional subspaces.

5.2.1 Lattice Traversal

Finding all interesting one-of-the-few claims requires computing the top-τ skyband in every non-empty subspace. We now describe, on a high level, how to accomplish this task using either Progressive or OnePass as the building block.

For Progressive, computation in every subspace starts with finding the skyline. For this part, we directly apply the techniques of Pei et al. (2006), who study the problem of computing the skyline in every subspace. Their techniques traverse the lattice of subspaces in either bottom-up or top-down order, and try to share computation across subspaces. For a given subspace $\mathcal{B}$, once we have computed the skyline using these techniques, we proceed with Progressive to compute the rest of the top-τ skyband as discussed in Section 5.2.

For OnePass, we traverse the lattice of subspaces top-down, i.e., going from low- to high-dimensional subspaces. We use the top-down technique from Pei et al. (2006) to help compute the skyline in a subspace using those found in its parent subspaces. Moreover, we use the top-τ skybands in parent subspaces to fine-tune the safe pro-

cessing order in the current subspace.

Furthermore, during a top-down lattice traversal, we use two tests to prune uninteresting high-dimensional subspaces from the search. 1) If the "distinct-count" of skyline objects in $\mathcal{B}$ (i.e., the number of distinct projections onto $\mathcal{B}$) is greater than τ, all descendants of $\mathcal{B}$ must have empty top-τ skybands. 2) Given $\mathcal{B}$, if the union of skyline objects from $\mathcal{B}$'s parents already contains more than τ skyline objects in $\mathcal{B}$, $\mathcal{B}$ must have an empty top-τ skyband.

5.3 Ranking Objects

This section presents our solution for ranking the set $\mathcal{O}$ of objects with attributes $\mathcal{A}$, based on what one-of-the-few claims can be made about them across subspaces, and how interesting these claims are. Ch. 5.3.1 proposes a novel scoring scheme that captures varying user preferences with a single parameter, while Ch. 5.3.2 describes how the algorithms in Ch. 5.2 can be leveraged to compute top-ranked objects.

5.3.1 Scoring Objects

A common approach for ranking a set $\mathcal{O}$ of objects (e.g., political candidates) is to combine multiple ranked lists of them (e.g., by voters). Traditionally, the scores for objects in a single list are assigned according to a *positional scoring vector (or function)* (Young, 1975), v, which maps a rank i ($1 \leqslant i \leqslant |\mathcal{O}|$) to a numeric score $v(i)$, such that $v(i) \geqslant v(i+1)$. Some examples include *Borda*, where $v(i) = |\mathcal{O}| - i$, and *Plurality*, where $v(1) = 1$ and $v(i) = 0$ for $i > 1$. Then, the overall object ranking is done according to the aggregate score of each object, usually defined as the sum of its scores across all ranked lists.

A straightforward approach would be to have one ranked list of $\mathcal{O}$ by each attribute in $\mathcal{A}$, and sum up an object's scores across these lists. However, this approach has serious drawbacks, particularly in handling data with correlation. Suppose o_1 is

ranked high for two correlated attributes (e.g., contributions from finance and real estate sectors, or minutes played and points scored), while o_2 is ranked equally high in two anti-correlated attributes (e.g., contributions from oil companies and environmental groups, or rebounds and assists). Since it is harder to rank high for two anti-correlated attributes, we should score o_2 higher than o_1 overall. However, the approach of summing scores over individual attributes will assign the same score to o_1 and o_2 (assuming all other factors are equal).

Aggregate Positional Scoring with Ties (APST) To resolve the issue above, we propose *Aggregate Positional Scoring with Ties*. The key novelty is to aggregate object scores not just across individual attributes, but instead over all non-empty subspaces. The dominance relationship in a multi-dimensional subspace $\mathcal{B}$ likely does not induce a totally ranked list; hence, we draw insight from Ch. 5.2 to score objects using the partial order in a way that reflects the uniqueness of one-of-the-few claims in $\mathcal{B}$. The result is a scoring scheme that naturally adapts to data distributions.

Definition 8 (Aggregate Positional Scoring with Ties (APST))**.** For each non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, sort $\mathcal{O}$ in ascending order of $\delta_{\mathcal{B}}(\mathcal{O}, o)$, the size of the dominating subset for each $o \in \mathcal{O}$; ties are broken arbitrarily. Denote this ranking by $\pi_{\mathcal{B}}$, where $\pi_{\mathcal{B}}(o) \in [1, |\mathcal{O}|]$ is the rank of o in this ranking. Let $[\pi^{\vdash}_{\mathcal{B}}(o)\,, \pi^{\dashv}_{\mathcal{B}}(o)]$ denote the range of ranks occupied by objects that tie with o in $\delta_{\mathcal{B}}(\mathcal{O}, o)$; i.e., $\pi^{\vdash}_{\mathcal{B}}(o) = \min\{\pi_{\mathcal{B}}(o') \mid \delta_{\mathcal{B}}(\mathcal{O}, o') = \delta_{\mathcal{B}}(\mathcal{O}, o)\}$ and $\pi^{\dashv}_{\mathcal{B}}(o) = \max\{\pi_{\mathcal{B}}(o') \mid \delta_{\mathcal{B}}(\mathcal{O}, o') = \delta_{\mathcal{B}}(\mathcal{O}, o)\}$. Given a positional scoring function v where $v(1) > v(2) > \cdots > v(|\mathcal{O}|)$, the *(APST) score of object o in $\mathcal{B}$*, denoted $\gamma_{\mathcal{B}}(o)$, is given by:

$$\gamma_{\mathcal{B}}(o) = \frac{1}{\pi^{\dashv}_{\mathcal{B}}(o) - \pi^{\vdash}_{\mathcal{B}}(o) + 1} \cdot \sum_{i \in [\pi^{\vdash}_{\mathcal{B}}(o), \pi^{\dashv}_{\mathcal{B}}(o)]} v(i).$$

Overall, the *APST* score of o, denoted $\Gamma(o)$, is the total of o's score over all non-empty subspaces: $\Gamma(o) = \sum_{\mathcal{B} \subseteq \mathcal{A}, \mathcal{B} \neq \varnothing} \gamma_{\mathcal{B}}(o)$.

Intuitively, for each subspace $\mathcal{B}$, APST sorts objects by tiers, and each tier occupies a range of ranks. The total score assigned by v to such a range of ranks is distributed equally among objects in the corresponding tier, as we consider them ties in $\mathcal{B}$. The larger the tier, the less "unique" each object, and the smaller the share each object will receive. A number of desirable properties follow directly from the definition.

- Suppose two tiers (in different subspaces) have sizes $\Delta < \Delta'$, but the ranges of ranks they occupy start from the same position. APST scores objects in the smaller tier (of size Δ) higher than those in the other tier, in their respective subspaces.

- Suppose two tiers (in different subspaces) are of the same size, but occupy different ranges of ranks starting from r and r', respectively, where $r < r'$. APST scores objects in the tier occupying the earlier range (starting with r) higher than those in the other tier, in their respective subspaces.

- Consider two tiers in the same subspace, where the range of ranks occupied by the first tier precedes that occupied by the second. APST scores objects in the first tier higher than those in the second, in this subspace.

Furthermore, aggregating scores over combinations of attributes make APST naturally adaptive to correlations in data. Consider again the example earlier in this section, where o_1 is ranked high for two correlated attributes $\{A_1, B_1\}$ and o_2 is ranked equally high in two anti-correlated attributes $\{A_2, B_2\}$. Since it is rare to rank high for anti-correlated attributes, few objects will be in o_2's tier or an earlier tier in $\{A_2, B_2\}$; therefore, APST will score o_2 high in $\{A_2, B_2\}$. On the other hand, for o_1, since A_1 and B_2 are correlated, there will likely be more objects dominating

o_1 in $\{A_1, B_1\}$ than those dominating o_2 in $\{A_2, B_2\}$. Thus, all other factors being equal, APST will score o_2 higher than o_1 overall.

Adjustable Discounted APST (APST-$\boldsymbol{\alpha}$) What remains to be discussed is the choice of the positional scoring function v. To make our scoring scheme easy to use, we have so far consciously avoided introducing any tuning parameters. However, as motivated in Example 6, we would like some degree of customization for ranking "specialized" objects relative to "well-rounded" ones. Recall that a specialized object is exceptional in very few attributes (such as Stockton in assists), while a well-rounded object is exceptional in none, but reasonably good in many, so as to be exceptional when all those attributes are combined (such as Bird). To capture user's wide ranging preferences for one or the other, we design our positional scoring function with a single tunable parameter to achieve that flexibility while retaining APST's desirable properties.

Definition 9 (Adjustable Discounted APST (APST-α))**.** Let α be a real number in $(0, 1)$. The *adjustable discounted APST (APST-α) score* of an object $o \in \mathcal{O}$ is o's APST score with positional scoring function $v_\alpha(i) = \alpha^{i-1}$.

Intuitively, the discounting factor, α, controls how much more higher ranked objects weigh against lower ranked objects, thereby affecting how specialized and well-rounded objects score relatively overall. In a higher-dimensional subspace $\mathcal{B}$, tiers tend to be larger. Hence, an object that is dominated by few others in $\mathcal{B}$ (but by more objects in subsets of $\mathcal{B}$) tend to score lower in $\mathcal{B}$, compared with an object with the same number of dominating objects in a lower-dimensional subspace $\mathcal{B}'$. With a smaller α, the score gap is wider, so overall, it is more difficult for well-rounded objects to surpass specialized ones, whose scores come mostly from contributions by low-dimensional subspaces. In Ch. 5.4, we will see how effective α is as a tuning

knob on real data—compared with weighted-sum (discussed in Ch. 5.1), APST-α has only 1 knob instead of d, but is able to produce comparable rankings as highly tuned weighted-sum.

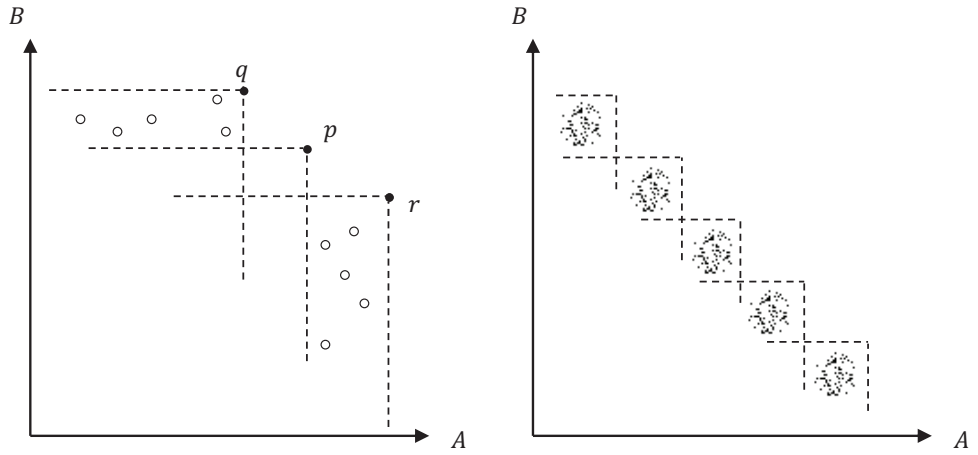
Comparison with Kemeny Optimal Rank Aggregation In addition to flexible approaches where users are allowed to specify their preferences, studies in social choice theory have also been investigating an alternative approach where some notion of "optimality" is defined so that the resulting ranks achieve that optimality. A popular representative of this approach is *Kemeny (optimal rank aggregation)*. As discussed in Ch. 5.1, given a set of rankings for $\mathcal{O}$, a Kemeny optimal rank aggregation is a ranking that minimizes the total number of pairwise disagreements between this ranking and the input rankings.

We give the formal definition below for completeness.

Definition 10 (Kendall Tau Distance, Kemeny Optimal Aggregation)**.** A ranking π of $\mathcal{O}$ is a linear ordering of objects in $\mathcal{O}$, and $\pi(o) \in [1, |\mathcal{O}|]$ denotes the rank of o. The *Kendall tau distance* between two rankings π and π', denoted $K(\pi, \pi')$, is defined as the number of pairwise disagreements between π and π'; i.e., $K(\pi, \pi') = |\{(o, o') \in \mathcal{O} \times \mathcal{O} \mid \pi(o) < \pi(o') \wedge \pi'(o) > \pi'(o')\}|$.

Given a set of d rankings $\pi_1, \ldots, \pi_d$ of $\mathcal{O}$, a Kemeny optimal aggregation is a ranking that minimizes the total Kendall tau distance to each of the d given rankings, i.e., $\arg\min_\pi \sum_{i=1}^{d} K(\pi, \pi_i)$.

It seems natural to apply Kemeny to the d rankings according to the individual attributes in $\mathcal{A}$. However, there are several issues. First, finding a Kemeny optimal aggregation is NP-hard in $|\mathcal{O}|$ (Dwork et al., 2001). We would prefer an approach that is computationally more tractable. Indeed, ranking objects using APST-α has complexity polynomial in $|\mathcal{O}|$.



(a) (b)

FIGURE 5.2: Kemeny fails to recognize a worthy object (p) but APST-α succeeds.

Second, by definition, Kemeny is strongly biased against specialized objects, which rank high in very few attributes but low in many, because ranking them high overall will incur many disagreements. As discussed in Ch. 5.1 following Example 6, based on points, rebounds, and assists per game, Kemeny would rank John Stockton low, even though he is a Hall-of-Famer who has the second highest assists per game in NBA history. In contrast, with a small α, APST-α would rank it high, because it likely lies on a small skyband for roughly half of the $2^d - 1$ non-empty subspaces—namely those containing the attribute in which the object specializes.

Third, Kemeny sometimes fails to suggest a rank for a worthy object p. More precisely, there may be many consensus rankings that are all optimal in the Kemeny sense, where p is ranked differently, sometimes below objects that can be considered obviously inferior to p. We illustrate this issue with an example below, and show how APST-α avoids this issue.

Example 7. Figure 5.2a depicts a set $\mathcal{O}$ of $2m + 1$ objects with two attributes A and B. Three most worthy objects, p, q, and r, are on the two-dimensional skyline;

q and r each dominate m objects, while p dominates none. Although p is mediocre in either A or B, no object dominates p in both A and B, and p's balance between A and B cannot be offered by any other object.

For Kemeny, note that any object that dominates p in A (or B) is dominated by p in B (or A, respectively). Since p represents the median in both A and B, no matter where p is placed in the consensus ranking, it induces the same number of disagreements ($2m$). Thus, p can be ranked anywhere in a Kemeny optimal aggregation.

Under APST-α, the scores of p, q, and r are:

$$\Gamma(p) = \gamma_{AB}(p) + \gamma_A(p) + \gamma_B(p) = (1 + \alpha + \alpha^2)/3 + 2\alpha^m;$$
$$\Gamma(q) = \Gamma(r) = (1 + \alpha + \alpha^2)/3 + 1 + \alpha^{m+1}.$$

To bound the rank of p according to APST-α, let k denote the number of objects in $\mathcal{O}\backslash\{p, q, r\}$ with score no less than $\Gamma(p)$. The sum of their scores is at least $k\Gamma(p)$ but obviously at most $\sum_{o\in\mathcal{O}\backslash\{p,q,r\}} \Gamma(o)$. Hence, $k \leqslant \sum_{o\in\mathcal{O}\backslash\{p,q,r\}} \Gamma(o) / \Gamma(p)$. The rank of p is no worse than:

$$\begin{aligned} k + 3 &\leqslant \textstyle\sum_{o\in\mathcal{O}\backslash\{p,q,r\}} \Gamma(o) / \Gamma(p) + 3 \\ &= \left(\textstyle\sum_{o\in\mathcal{O}} \Gamma(o) - (\Gamma(p) + \Gamma(q) + \Gamma(r))\right) / \Gamma(p) + 3 \\ &= \tfrac{3(1-\alpha^{2m+1})/(1-\alpha) - (1+\alpha+\alpha^2+2+2\alpha^m+2\alpha^{m+1})}{(1+\alpha+\alpha^2)/3+2\alpha^m} + 3 \\ &\approx 3 \cdot \frac{1 + 2\alpha}{1 - \alpha^3}. \end{aligned}$$

The simplification in the last step above is obtained by ignoring α^{2m+1}, α^m, and α^{m+1}, assuming that m is sufficiently large.

Hence, with a small enough α, APST-α can rank p as high as the third—better than all other objects except q and r.

This example can be generalized to a case consisting of multiple object clusters where Kemeny fails while APST-α succeeds (Figure 5.2b). Suppose any two objects

from different clusters do not dominate each other. For Kemeny, objects from different clusters are indistinguishable, by the same argument for the simple case above. Thus, it is possible for a skyline object in some cluster to rank lower than all objects from other clusters. In contrast, APST-α favors skyline objects in each cluster, especially those close to either the A or B axis.

Finally, Kemeny does not allow for customization—if it fails to recognize a worthy object there is little that a user can do. On the other hand, the discounting factor α in APST-α provides a single, effective knob that allows users to choose their preference between specialized and well-rounded objects. Indeed, we have seen from above how the choice of α helps overcome the issues faced by Kemeny. More detailed results on real data are shown in Ch. 5.4.

5.3.2 Finding Top κ Objects

Computing the exact APST-α scores of all objects in subspace entails computing $\delta_{\mathcal{B}}(\mathcal{O}, o)$ for all $o \in \mathcal{O}$ and non-empty $\mathcal{B} \subseteq \mathcal{A}$, which takes $O(2^d|\mathcal{O}|^2)$ time using a naive algorithm. However, for the purpose of identifying objects most worthy of further investigation, we care only about the top-ranked objects. This observation leads to the following problem definition:

Definition 11 (Finding Top κ Objects)**.** Given a set $\mathcal{O}$ of objects, a discounting factor $\alpha \in (0, 1)$, and $\kappa \geqslant 1$, find the top κ objects in $\mathcal{O}$ ranked by APST-α scores.

We show how to extend the algorithms in Ch. 5.2 to efficiently compute approximate answers to the above question. The key observation is that, in each subspace, we need to compute only enough number of tiers in order to not miss any top objects overall, because score contributions from memberships in subsequent tiers are so small that they have little influence over whether an object can be in the top κ overall.

Given an error allowance ϵ, we can compute a list of objects, where each object o gets an approximate score $\tilde{\Gamma}(o) \in (\Gamma(o) - \epsilon, \Gamma(o)]$. We divide the error allowance ϵ evenly among the subspaces to be considered. Within a subspace $\mathcal{B}$ with share $\epsilon_{\mathcal{B}}$ of error allowance, we use Progressive or OnePass to "grow" the skyband up to some top-τ skyband such that any object outside it will receive a score below $\epsilon_{\mathcal{B}}$ in $\mathcal{B}$. We add the objects in this skyband to the output list (or update their scores if they are already in it). With some care, we can ensure that OnePass retains its advantage over Progressive; i.e., it avoids computing the entire "next" tier, which could include all remaining objects. The idea is that we only need to see enough number of objects in this tier before knowing that all its objects must score below $\epsilon_{\mathcal{B}}$, because APST scores get "diluted" by larger tiers. When we are done with $\mathcal{B}$, we can derive a tighter bound (hopefully much less than $\epsilon_{\mathcal{B}}$) for errors in $\mathcal{B}$, and use it to update the error allowance for remaining subspaces.

The procedure described is formally presented as follows. $N_{\mathcal{A}}$ denotes the number of subspaces that have not been visited. $\hat{\epsilon}$ denotes the maximum possible error accrued so far.

1. Initialize $N_{\mathcal{A}} = 2^d - 1, \hat{\epsilon} = 0$.

2. For each subspace:

   2.1 $\tau \leftarrow \lceil \log_\alpha((\epsilon - \hat{\epsilon})/N_{\mathcal{A}}) \rceil$

   2.2 Compute the top-τ skyband and the next tier if the top-τ skyband contains less than τ objects.

   2.3 Suppose the next tier starts at $x + 1$ and ends at $x + \Delta$, where $x < \tau < x + \Delta$. If $\alpha^x \cdot \frac{1-\alpha^\Delta}{(1-\alpha)\cdot\Delta} > (\epsilon - \hat{\epsilon})/N_{\mathcal{A}}$, output this tier together with the top-τ skyband and update $\hat{\epsilon} \leftarrow \hat{\epsilon} + \alpha^{x+\Delta}$. Otherwise, ignore the last tier for score aggregation, and update $\hat{\epsilon} \leftarrow \hat{\epsilon} + \alpha^x \cdot \frac{1-\alpha^\Delta}{(1-\alpha)\cdot\Delta}$

2.4 $N_{\mathcal{A}} \leftarrow N_{\mathcal{A}} - 1$.

Now we show that step 2.2 above can be done by extending either Progressive or OnePass .

Extending Progressive We change Progressive so that it computes not the largest at most τ, but the smallest skyband that contains at least τ objects. (removing line 6 and while condition be $|\Phi| \geqslant \tau$). This change does not affect the running time of Progressive , because progressive compute the next tier anyway.

Extending OnePass Suppose in each iteration, the last tier computed so far starts at $x+1$ and ends at $x+\Delta$. We change the condition on line 9 to $\alpha^x \cdot \frac{1-\alpha^\Delta}{(1-\alpha)\cdot\Delta} \leqslant (\epsilon - \hat{\epsilon})/N_{\mathcal{A}}$. The worse case running time of OnePass is then modified to $O(ny)$, where y is the smallest integer such that $\frac{1-\alpha^y}{(1-\alpha)\cdot y} < \epsilon/(2^d - 1)$.

The user does not need to choose ϵ manually. A reasonable default is $\alpha^{\kappa-1}$, the value of the positional scoring function at rank κ. With this setting, we can guarantee that any object not in the output list cannot be among the top κ overall. The output list may contain more than κ objects, and ϵ can be used to determine which part of this ranking is guaranteed to be accurate. As future work, we are developing an "online" version of the algorithm where ϵ is incrementally tightened when given additional time or until the top κ objects can be accurately separated from the rest.

5.4 Experiments

5.4.1 Datasets

All algorithms were implemented in C++. We conducted all experiments on a machine with Intel Core i7-2600 3.4GHz processor and 7.8GB memory. We used the following real and synthetic datasets.

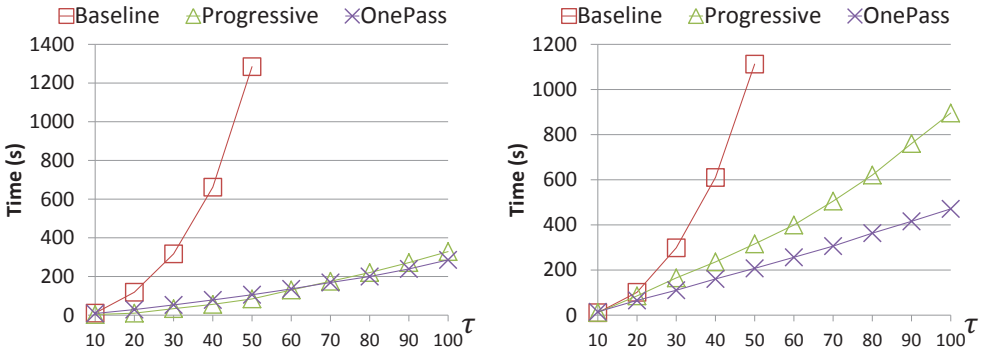
Real datasets

- *NBA players' career total statistics (NBA1)* [2]: This dataset contains the total career performance of $\sim 4K$ players who have played in NBA ($n \approx 4K$), where the performance is captured by 15 numeric attributes ($d = 15$) including # games played, points, rebounds, etc.
- *NBA players' career average statistics (NBA2)*: This dataset captures the career average performance of each player, i.e., the attribute values for each player are derived by dividing the corresponding values in the above NBA1 dataset by the number of games played by the player. Hence it also has $\sim 4K$ players ($n \approx 4K$) and all the numeric attributes in NBA1, except # games played ($d = 14$).
- *NBA players' game-by-game statistics (NBA3)*: This dataset contains the performance of each player in each game, with in total $\sim 400K$ individual performance records ($n \approx 400K$) on 14 numeric attributes ($d = 14$).
- *National Research Council survey data on 127 Computer Science programs (NRC)* [3]: This dataset contains the assessment of 127 CS programs in the US ($n = 127$) on 20 numeric attributes ($d = 20$). The data was used by NRC to rank these programs.

Synthetic datasets: We used three types of synthetic datasets– correlated (CORR), anti-correlated (ANTICORR), and independent (IND), under varying dimensionality d, dataset cardinality n, and skyband size τ. CORR and ANTICORR were generated in the following way. We randomly sampled points by multi-variate Gaussian distri-

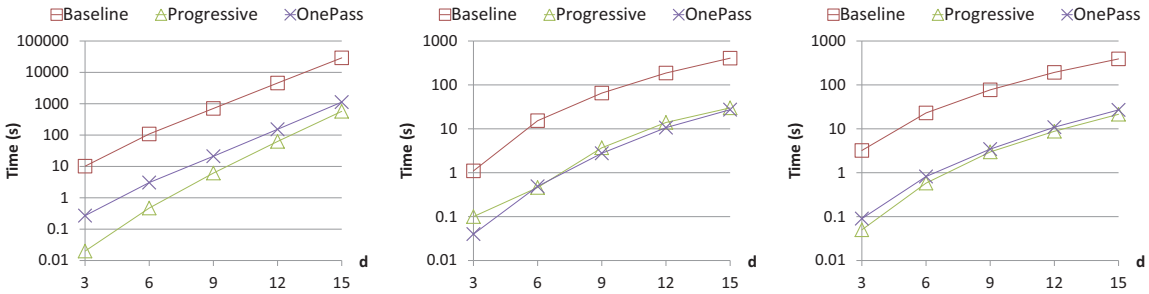
[2] NBA datasets were from `http://www.databasebasketball.com/`.

[3] The NRC data is from `http://www.nationalacademies.org/nrc/`.



(a) NBA1 (b) NBA3

FIGURE 5.3: Algorithm elapsed time on NBA1, NBA3 (varying τ)



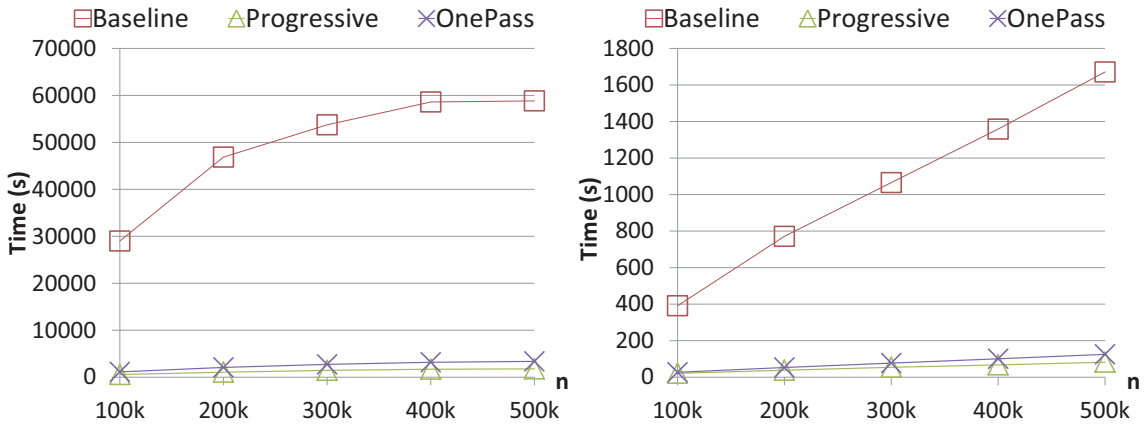
(a) CORR (b) ANTICORR (c) IND

FIGURE 5.4: Elapsed time of algorithms on synthetic data (varying d)

bution. We then stretched all the points in the direction of $(1, 1, \ldots, 1)$ to produce CORR, and we shrunk them in the direction of $(1, 1, \ldots, 1)$ to yield ANTICORR. This way, all the attributes are pairwise correlated or anti-correlated.

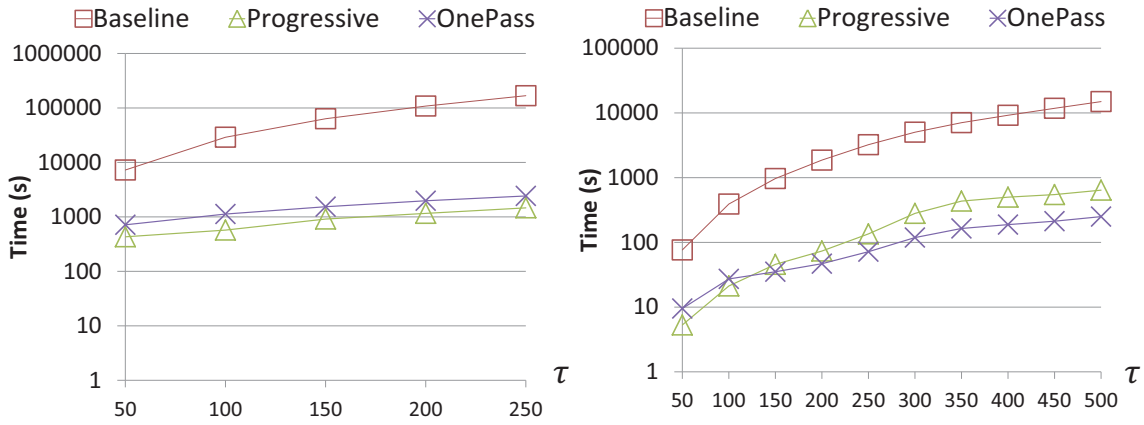
5.4.2 Efficiency of Top-τ Skyband Algorithms

Given a dataset with d numeric attributes, a particular τ value, and an algorithm, we used the algorithm to find the top-τ skyband for each and every subspace. The total elapsed time for the 2^d-1 nonempty subspaces is used to measure the algorithm's efficiency. We compared the efficiency of Baseline, Progressive (Algorithm 9), and OnePass (Algorithm 10). Baseline was also introduced in Ch. 5.2. It iteratively computes k-skyband using a simple extension of the BUS algorithm in Pei et al.



(a) CORR (b) IND

FIGURE 5.5: Elapsed time of algorithms on synthetic data (varying n)



(a) CORR (b) IND

FIGURE 5.6: Elapsed time of algorithms on synthetic data (varying τ)

(2006), from $k = 1$ to the k value such that the corresponding k-skyband contains at least τ objects.

Efficiency Experiments on Real Datasets Figure 5.3 shows the execution elapsed time of Baseline, Progressive and OnePass, under varying $\tau = 10, 20, \ldots, 100$, on both the 4K-tuple NBA1 dataset and the 400K-tuple NBA3 dataset. On both the small and large datasets, our algorithms significantly outperformed the baseline approach.

On the small NBA1 dataset (Figure 5.3a), OnePass was less efficient for small τ and started to outperform Progressive as τ increased. This is because the dimensionalities of subspaces in which top-τ skyband is empty but cannot be pruned (by techniques in Ch. 5.2.1) increase as τ increases. Computing the skylines for these subspaces gets more expensive as SUBSKY becomes less efficient.

On the large NBA3 dataset (Figure 5.3a), we also observe the running time of Progressive grew faster than OnePass as τ increased, due to the same reason for Figure 5.3a. Besides, attributes in NBA3 have smaller correlations than NBA1 attributes, which induces larger skylines for the same subspace and makes Progressive even less efficient. This is why OnePass already outperforms Progressive when τ is small.

Efficiency Experiments on Synthetic Datasets Our experiments on the synthetic datasets CORR/ANTICORR/IND verified the scalability of Progressive and OnePass.

Varying d, fixed n=100,000 and τ=100: Figure 5.4 shows that both Progressive and OnePass were faster than Baseline by orders of magnitude (vertical axis in logarithmic scale). Progressive run faster than OnePass on (a) correlated data. Their performances were comparable on (b) anti-correlated and (c) independent data. We observe that the elapsed time on CORR is much longer than that on ANTICORR and IND for all three algorithms. This is because the top-τ skyband in CORR may be non-empty even in high dimensional subspaces, while for ANTICORR and IND, top-τ skyband for a small τ is empty in most subspaces. Such cases can be quickly detected and pruned as discussed in Ch. 5.2.1. When the dimensionality is high, due to the way ANTICORR was generated for ensuring pairwise anti-correlation among all attributes, ANTICORR becomes similar to IND. So we omit the results on ANTICORR in the following discussion.

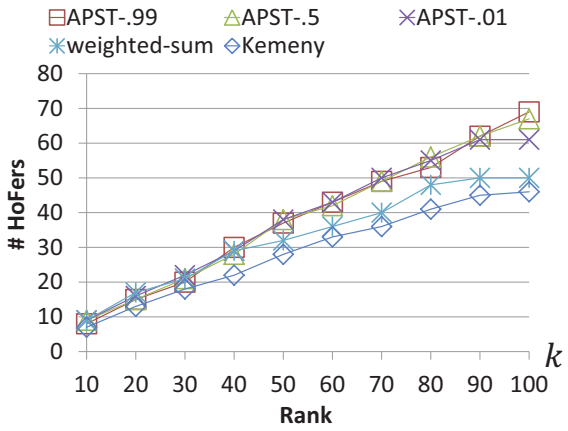


FIGURE 5.7: Comparison of rankings by number of HoFers in top-k

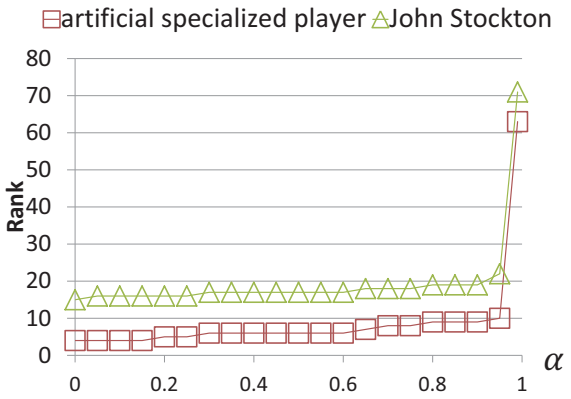


FIGURE 5.8: Rank of artificial player with different α

Varying n, fixed d=15 and τ=100: From Figure 5.5, we observe again that Progressive and OnePass outperformed Baseline by orders of magnitude, and Progressive slightly outperformed OnePass. Their elapsed time increased roughly linearly in n.

Varying τ, fixed n=100,000 and d=15: Figure 5.6 shows that Progressive and OnePass significantly outperformed Baseline on both CORR and IND. Their elapsed time grew nearly exponentially by τ for all three algorithms (vertical axis in logarithmic scale). Progressive was slightly faster than OnePass on CORR. On IND dataset, OnePass was slower than Progressive under small τ but became faster than the latter as τ increased.

5.4.3 Quality of Ranking by APST-α

We evaluated the quality of the rankings produced by APST-α and Kemeny on the NBA and NRC datasets, and weighted-sum ranking on NBA.

APST-α vs. Weighted-Sum on NBA Dataset

In this experiment, we used APST-α and weighted-sum to rank NBA players by their career average performance (NBA2). We varied α to produce different APST-

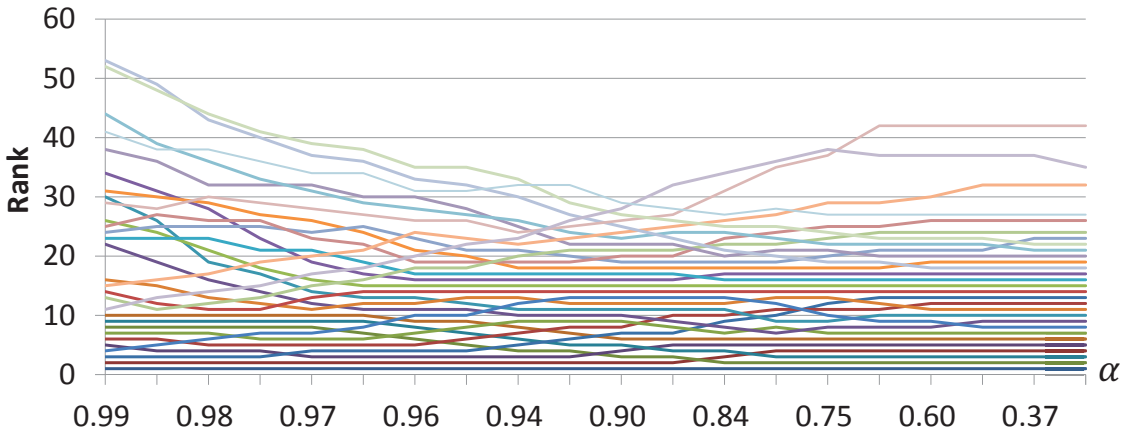


FIGURE 5.9: How the APST-α ranks of NBA players change by α

α rankings. For each ranking results, we measure its quality by the number of Hall of Fame inductees (HoFers for short) in the top-k players according the the ranking, under various k values.

In NBA2, 7 out of the 14 attributes are used for ranking because statistics such as steals, blocks, and minutes were not recorded for games played before 1971 in the dataset. Thus, including these attributes would underrate the HoFers from the early days. Moreover, we did not use career total performance data (NBA1) in this experiment because that will also underrate earlier HoFers, as they did not play as many games as recent players.

For weighted-sum, the weights on the attributes were determined as follows. We first found a linear classifier to separate HoFers from non-HoFers that minimizes the number of inaccurately classified players. We then used the (normalized) vector perpendicular to the linear classifier as the weight vector in weighted-sum ranking.

The result of this experiment is shown in Figure 5.7. It compares eighted-sum, APST-.99, APST-.5, and APST-.01, by the numbers of HoFers in the top-k of their rankings, respectively. We can see that APST-α contains visibly more HoFers in top-k under most considered combinations of α and k. Note that the performance of

all ranking methods were negatively affected by the existence of active players who are future HoFers but not eligible for induction until 5 years after retirement. For the sake of accuracy, these players were not considered in obtaining the weight vector but were included in ranking.

APST-α vs. Kemeny on NBA Dataset

We also used Kemeny for the same experiment described above, and the result is also shown in Figure 5.7. We can see that in most cases APST-α rankings contain visibly more HoFers in top-k than Kemeny ranking. However, we should note that APST-α and weight-sum rankings were produced for all 4K players in NBA2, while we ranked only 200 NBA players for Kemeny. There are 91 HoFers in the 4K players in NBA2 and 59 of them are in the 200 players for Kemeny. The 200 players is the set union of 7 lists on the 7 attributes participated in ranking, where each list contains the top-40 players on the corresponding attribute.

We used only 200 players due to practical reasons as follows. To compute Kemeny optimal rank aggregation, we modeled it as an Integer Programming (IP) problem and used IBM CPLEX Optimizer ([4]) to solve it. CPLEX failed to find solutions when there are more players due to precision in computation. Moreover, finding Kemeny optimal is known to be NP-hard under 4 or more attributes Dwork et al. (2001). The corresponding IP problem is also memory-intensive as it requires $O(n^3)$ variables for n objects (i.e., NBA players). Hence CPLEX caused memory overflow with close to 300 players.

For fairness in comparison, we also used APST-α to rank the 200 objects only, in which it showed comparable performance to Kemeny.

[4] `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

5.4.4 Effect of α in APST-α

To better understand the parameter α in APST-α and investigate how it is related to promoting specialized or well-rounded objects, we injected an artificial player into the NBA1 and NBA2 datasets. Again, we used 7 (instead of all) attributes, as mentioned in Ch. 5.4.3, to avoid underrating earlier players.

The artificial player has the highest value on attribute points but the lowest values in all other attributes. Figure 5.8 shows how this player's rank changed in APST-α ranking by varying α. As α decreased from 1, the artificial player quickly got ranked high among real players. Its rank converged as α approached 0, but never became the first because it cannot surpass the players that have the highest values on some other attributes. Similar observation was made when the artificial player is specialized in other attributes. This result agrees with the discussion in Ch. 5.3 that a smaller α favors those objects that specialize in a few dimensions over well-rounded players. An example of specialized player is John Stockton, a HoFer, who is second only to Magic Johnson in assists per game, but is not ranked high in any other considered attributes. His ranking positions under various α values are also plotted in Figure 5.8, and his curve shows similar trend to that for the artificial player.

In Figure 5.9, we focus on the top-30 players by APST-.9 on NBA2, under the three most frequently used attributes in assessing player performance– points, rebounds and assists. The figure shows how their ranking positions change by α (horizontal axis in logarithmic scale). At the bottom of the figure, the top-9 players always remain in top-13, due to their exceptional performance in multiple attributes. The ranks of the remaining players change drastically by α. These players can be mainly categorized into two types: I) specialized players, who are exceptional on very few attributes; II) well-rounded players, who are not exceptional on any single attribute, but are reasonably good on multiple attributes and hence are exceptional with re-

gard to those attributes combined. In Figure 5.9, the curves of Type-I players go down as α decreases, because small α favors them more against other players in at least half of the subspaces that contain the attribute on which they specialize. On the contrary, the curves of Type-II players go up as α decreases. They get rewarded in higher dimensional subspaces for not being dominated by many other players in such subspaces. However, they can be easily dominated by others on individual dimensions.

5.5 Related Work

Our work is closely related to skyline computation for multidimensional data, which can roughly be grouped into two classes. The first class (Börzsönyi et al., 2001; Chomicki et al., 2003; Godfrey et al., 2005) include Block Nested Loop (BNL), Divide and Conquer (D&C) and their variants, and do not require precomputed indices. The second class of algorithms (Tan et al., 2001; Kossmann et al., 2002; Papadias et al., 2005) adopt B^+-tree and R-tree indices to prune away unnecessary computation. None of those considers all subspaces or k-skybands. Skyline computation for multiple (or all) subspaces are first studied in Pei et al. (2006) (BUS and TDS algorithms) and in Tao et al. (2006) (SUBSKY algorithm). The former are extensions of BNL and D&C algorithms that traverse the subspace lattice without using indexing structured, while the latter follows the index-based approaches. Extending skyline to k-skyband was first studied in Papadias et al. (2005).

While our notion of top-τ skyband is the first to provide a universal parameter suitable for selecting the right set of objects regardless of the subspaces being considered, there are previous works on various advanced selection criteria when the size of the skyline is too large. For example, BBS (Papadias et al., 2005) and PBT (Yiu and Mamoulis, 2007) focus on selecting points that dominate the most other points, and Lin et al. (2007) focus on selecting a fixed sized set of points that collectively

dominate the most other points. More recently, Lu et al. (2010) propose the notion of layered skylines, where each layer is defined as the skyline after objects in previous layers are removed. Neither of those works consider the need for compatibility across all subspaces as we do.

Linear ranking of objects for the purpose of answering top-k query have been studied extensively, including those using skyline and skybands Tsaparas et al. (2003); Vlachou et al. (2008). Those studies focus on performance improvements and do not consider providing better control of the semantics of the ranking functions. Kemeny ranking (Dwork et al., 2001), on the other hand, was proposed in social choice theory to achieve good semantics. It aims to minimize the disagreements between the aggregated ranking and the input rankings, a problem turns out to be NP-hard (if more than 3 rankings are to be aggregated). While useful, Kemeny does not provide a flexible knob, such as our α parameter, for adjusting the ranking based on the users' preference towards different types of prominent objects.

5.6 Conclusion

The tasks of finding one-of-the-few claims from data and ranking objects by such claims are important to the nascent field of computational journalism. We have introduced a uniqueness-based measure of interestingness, which leads to a simple and intuitive problem formulation for finding all interesting claims, using a single uniqueness threshold τ that automatically adapts to data characteristics and applies to all subspaces. We have proposed a novel scheme for ranking objects, overcoming the inflexibility of Kemeny without resorting to a large number of knobs like weighted-sum. We have devised efficient algorithms for both tasks, using techniques such as pruning and approximation to tame complexity. We have seen promising results on real data, and we believe that our attention to usability will appeal to journalists and citizens alike.

6

Visualization Assisted Exploration

6.1 Introduction

In previous chapters, we have discussed how to carry out fact-checking and lead-finding tasks within the QRS-based framework, with the key ingredient being measuring a claim's quality by comparing it against claims in the same form. As another example, consider the following two claims about the performance of two NBA players.

- *Kevin Love's 31-point, 31-rebound game on Friday night ... Love became the first NBA player with a 30/30 game since Moses Malone had 38 points and 32 rebounds in a game back in 1982.*[1]
- *He (LeBron James) scored 35 or more points in nine consecutive games and joined Michael Jordan and Kobe Bryant as the only players since 1970 to accomplish the feat.*[2]

[1] `http://espn.go.com/espn/elias?date=20101113`

[2] `http://www.nba.com/cavaliers/news/lbj_mvp_candidate_060419.html`

The purposes of both claims above are to highlight some player's performance, but they describe different aspects of the game. Common to both claims is the attempt to impress the reader by stating that few others have done the same or better before.

However, instead of just singling out outliers in text, it is more powerful to visualize the set of points representing all claims of the same form. Figure 6.1a shows one such visualization of all NBA players' `points` and `rebounds` stats in a single game by treating them as 2D points and plotting them in a scatter plot of the "sparse" points and a heatmap showing the density of the remaining points (we will define "sparse points" formally in Section 6.2.2). The visualization gives clear context on how impressive Kevin Love's 31/31 performance is by showing not only whether the performance is on the skyline, but also how far away it is from the edge of the cloud of mediocre performances. Furthermore, this single visualization can help the users explore many outstanding performances for which the same form of claims can be made, and see the distribution of players' performance in terms of `points-rebounds` in a single game. A similar visualization can be generated to evaluate LeBron James' 9-game 35-plus-point streak. Indeed, this visualization can be used for any form of claim that can be represented by a 2d scatter plot.

We identify two visual features essential to data exploration—outliers and clusters. This leads us naturally to the choice of overlaying a scatter plot (for outliers) on a heatmap (for clusters).

Given a large set $\mathcal{S}$ of points to visualize, we do not need to show the exact distribution of $\mathcal{S}$ because in practice because ordinary users are unable to perceive the difference between two dense regions containing a similar number of points (say 200 versus 210). Approximation can be easy in certain cases. For example, in the Kevin Love case, if the underlying data are stored as points and rebounds per player per game, each exploration query is a simple lookup. One can apply many existing

outlier detection and density-estimation techniques to produce an approximation of the final plot. In many other cases, however, computing this visualization, even approximately, is a non-trivial task since it involves running many potentially expensive aggregate queries over the entire database. For example, in the LeBron James case, it takes an algorithm linear time to scan through a player's game-by-game scoring stats to produce $\mathcal{S}$ representing his *prominent scoring streaks* (Jiang et al., 2011). In such cases, without knowing how the input is transformed into $\mathcal{S}$, it is impossible to sample directly from $\mathcal{S}$.

We observe that many datasets for exploration are comprised of data for objects (e.g., players or teams), and an exploration is often represented as a 2D point set $\mathcal{S}$ obtained by evaluating a blackbox exploration query on all the objects. It is clear that the objects do not contribute equally to the visual properties of the visualization of $\mathcal{S}$: many objects produce only points that will be buried inside dense regions, and are not interesting from the visualization perspective except that they contribute to the density color.

Consider a user-provided exploration query modeled as a function f that maps the data of each object to a set of 2d points, our goal is to, without knowing how f behaves, find a small set of objects, evaluate f on their data and produce a good approximation of the scatter plot with heatmap visualization of $\mathcal{S}$ and preserve the two aforementioned key visual features: outliers and clusters.

The main contributions of this chapter include:

1. We formally define the two key visual features of scatter plot with heatmap type visualization, and quantify the quality of approximation.

2. We propose a two-phase sampling-based algorithm that efficiently generates an approximation of $\mathcal{S}$ for visualization without obtaining the exact $\mathcal{S}$ by evaluating f on the full data. Quantitative justification is provided.

Table 6.1: Table of notations.

Notation	Description
$\mathcal{R}_i$	data for object i, as a sequence of tuples
n_i	number of tuples in $\mathcal{R}_i$
N	number of objects
$\mathcal{D}$	dataset, as a collection of data $\mathcal{R}_{1\dots N}$ for objects $1 \dots N$
$\|\mathcal{D}\|$	total number of tuples in $\mathcal{D}$
f	an exploration query
$\mathcal{S}_f(\mathcal{D})$	result of evaluating f on $\mathcal{D}$, as a multi-set of 2d points
$\mathcal{N}_{\mathcal{R}}(p; r)$	neighborhood of a 2d point p given radius r
$\mathcal{N}_{\mathcal{S}}(p; r)$	neighbors of a 2d point p in multi-set $\mathcal{S}$ given radius r
$\mathcal{S}_{\text{sparse}}$	points of $\mathcal{S}$ in a sparse neighborhood
$\mathcal{S}_{\text{sketch}}$	a "sketch" of $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$, as a set of weighted 2d points
r_x, r_y	neighborhood size
τ	neighborhood density threshold for points in $\mathcal{S}_{\text{sparse}}$

3. We perform extensive experiments to evaluate both the quality of approximation and the efficiency of the sampling-based algorithm for interactive data exploration.

6.2 Exploration Query Problem

In this section, we introduce the basic notations (Section 6.2.1) and formally define the problem (Section 6.2.2). In Section 6.2.3, we present three examples of exploration queries that we study in this chapter. For readers' convenience, we summarize all notations in Table 6.1.

6.2.1 Preliminaries

Suppose we have data $\mathcal{D}$ for N distinct objects as N relations $\mathcal{R}_1, \dots, \mathcal{R}_N$. The data for object i, $\mathcal{R}_i = (t_{i,1}, \dots, t_{i,n_i})$, is an ordered set of n_i tuples. All tuples $t_{i,j}$ conform to the same schema R. Let $\|\mathcal{D}\| = \sum_{i=1}^{N} n_i$ denote the total number of tuples for all objects.

Let f denote an *exploration query* that takes input an ordered set of tuples conforming to R, and outputs a (multi-)set of points in $\mathcal{R}$. The result of evaluating

f on data $\mathcal{D} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$ is denoted by $\mathcal{S}_f(\mathcal{D})$, or simply $\mathcal{S}$ when the context is clear. $\mathcal{S}_f(\mathcal{D})$ is defined as the bag of the results of evaluating f on each $\mathcal{R}_i$, i.e., $\mathcal{S}_f(\mathcal{D}) = \uplus_{i=1}^{N} f(\mathcal{R}_i)$, or simply $\mathcal{S}$ when the context is clear. We formally define the *neighborhood* of a point $p \in \mathcal{R}$ as

$$\mathcal{N}_{\mathcal{R}}(p; r) = \{p' \in \mathcal{R} \mid \|p - p'\| \leqslant r\},$$

with the radius r as an input.

The *neighbors* of a point p in a finite set $\mathcal{S}$ is denoted by

$$\mathcal{N}_{\mathcal{S}}(p; r) = \mathcal{N}_{\mathcal{R}}(p; r) \cap \mathcal{S}.$$

In the remainder of the chapter, we use (scaled) L_∞-norm for $\|\cdot\|$ in the neighbor definition above, i.e.:

$$\mathcal{N}_{\mathcal{R}}(p; r_x, r_y) = \{p' \in \mathcal{R} \mid |p.x - p'.x| \leqslant r_x \wedge |p.y - p'.y| \leqslant r_y\};$$
$$\mathcal{N}_{\mathcal{S}}(p; r_x, r_y) = \mathcal{N}_{\mathcal{R}}(p; r_x, r_y) \cap \mathcal{S}.$$

However, the results of this chapter extend to other commonly used norms as well, e.g., L_1-norm, L_2-norm.

6.2.2 Problem Definition

Given an exploration query f on dataset $\mathcal{D}$, our goal is to efficiently generate the following two components for visualizing $\mathcal{S}_f(\mathcal{D})$.

- $\mathcal{S}_{\text{sparse}}$: points of $\mathcal{S}_f(\mathcal{D})$ in a sparse neighborhood (for scatter plot);
- $\mathcal{S}_{\text{sketch}}$: a "sketch" of rest of the points, as a set of weighted points (for heatmap).

We define the *sparse points* $\mathcal{S}_{\text{sparse}}$ as the set of points whose number of neighbors is no more than some *sparsity threshold* τ. Given r_x and r_y that define the neigh-

borhood of a point and sparsity threshold τ, all points with a sparse neighborhood can be precisely identified.

For all the other points, i.e., $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$, we create a *sketch* $\mathcal{S}_{\text{sketch}}$ (not to be confused with the concept of sketch in data streams). The sketch consists of a set of weighted points, where $(p, w_p) \in \mathcal{S}_{\text{sketch}}$ represents an estimated w_p points of $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$ in p's neighborhood. A good sketch should have each point of $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$ "covered" by one point of $\mathcal{S}_{\text{sketch}}$. We use the *sketch distance function* δ, which will be defined later in this section, to capture the quality of a sketch $\mathcal{S}_{\text{sketch}}$ measured w.r.t. $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$.

The challenge in evaluating an exploration query comes with a computation budget. To serve the purpose of realtime exploration, we would like to evaluate the query f without performing the evaluation on the full data of all objects. With a limited amount of data accesses, an approximate solution $(\tilde{\mathcal{S}}_{\text{sparse}}, \mathcal{S}_{\text{sketch}})$ needs to be found. The quality of $\tilde{\mathcal{S}}_{\text{sparse}}$ is measured by its precision and recall w.r.t. $\mathcal{S}_{\text{sparse}}$, while the quality of $\mathcal{S}_{\text{sketch}}$ is measured by δ w.r.t. $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$.

Motivated by achieving as high quality as possible for both $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ within a budget constraint, we formally define the exploration query evaluation problem as follows.

Definition 12 (Exploration Query Evaluation)**.** With a relational schema R, a dataset of N objects $\mathcal{D} = \{\mathcal{R}_i\}_{i=1}^{N}$, the full result of an exploration query f is given by $\mathcal{S} = \uplus_{i=1}^{N} f(\mathcal{R}_i)$.

Given neighborhood radius r_x, r_y, a sparsity threshold τ, solve the following two tasks:

- **Task 1.** Find $\tilde{\mathcal{S}}_{\text{sparse}}$ that approximates the set of sparse point $\mathcal{S}_{\text{sparse}} \subseteq \mathcal{S}$, where

$$\mathcal{S}_{\text{sparse}} = \{p \in \mathcal{S} \mid |\mathcal{N}_{\mathcal{S}}(p; r_x, r_y)| \leqslant \tau\}.$$

- **Task 2.** Find a weighted point set

$$\mathcal{S}_{\text{sketch}} = \{(p, w_p) \mid p \in \mathcal{S} \wedge w_p \in \mathbb{Z}^+\}$$

that minimizes $\delta(\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}, \mathcal{S}_{\text{sketch}})$.

subject to a given computation budget $\eta \in (0, 1]$ such that at most $\eta \cdot \|\mathcal{D}\|$ tuples can be accessed during evaluation.

Sketch Distance. The distance function δ measures the quality of the sketch $\mathcal{S}_{\text{sketch}}$ w.r.t. $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$ and needs to capture two aspects of sketch quality:

I. the *distribution* of $\mathcal{S}_{\text{sketch}}$ should resemble that of $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$;

II. the *magnitude* of $\mathcal{S}_{\text{sketch}}$ should be close to that of $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$.

Definition 13 (Sketch Distance)**.** Given a multiset of points $P = \{p_1, p_2, \ldots, p_n\}$ and a weighted point set $Q = \{(q_1, w_1), (q_2, w_2), \ldots, (q_m, w_m)\}$, the *sketch distance*[3] between P and Q is defined as

$$\delta(P, Q) = 1 - \frac{\mathsf{OPT}}{\max\{|P|, \|Q\|\}},$$

where $\|Q\| = \sum_{j=1}^{m} w_j$ and OPT is the optimal solution to the following integer program:

$$\text{maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \mathbf{1}[p_j \in \mathcal{N}_{\mathcal{R}}(q_i; r_x, r_y)], \tag{6.1}$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij} \leqslant w_i, \ 1 \leqslant i \leqslant m \tag{6.2}$$

$$\sum_{i=1}^{n} x_{ij} \leqslant 1, \ 1 \leqslant i \leqslant n \tag{6.3}$$

$$x_{ij} \in \{0, 1\}, \ 1 \leqslant j \leqslant n, \ 1 \leqslant i \leqslant m. \tag{6.4}$$

[3] The sketch distance is adapted from the Earth Mover's Distance (Rubner et al., 1998) widely used in measuring the dissimilarity between two images in image processing. Adaptations are made to suit the purpose of this work.

Consider P as the set of points to be sketched, i.e., $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$, and the weighted point set Q as the sketch. A point $(q, w) \in Q$ can cover a point $p \in P$ if p and q are neighbors, and q can cover at most w points in P. On the other hand, each point of p can be covered by at most one point of Q. Hence, OPT is the maximum number of points of P that Q can cover.

The quantity $\frac{\mathsf{OPT}}{\max\{|P|,\|Q\|\}}$ measures the similarity between P and sketch Q. Dividing by the larger of $|P|$ and $\|Q\|$ penalizes possible disparity between them.

Let us revisit Kevin Love's 31-point, 31-rebound game example. Consider the performance of all NBA players in all games in terms of point-rebound, and visualize all points ($\approx 10^6$ of them) by combining $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$.[4] In Figure 6.1a, points of $\mathcal{S}_{\text{sparse}}$ are plotted as red dots. $\mathcal{S}_{\text{sketch}}$ is visualized using heatmap, by distributing the weight of each point evenly in its neighborhood. Here, $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ are produced using an expensive baseline algorithm (Algorithm 11), which performs full evaluation on the entire dataset $\mathcal{D}$ and guarantees $\delta(\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}, \mathcal{S}_{\text{sketch}}) = 0$.

On the other hand, Figure 6.1b visualizes, in the same way, approximated sets $\tilde{\mathcal{S}}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ produced by Algorithm 12, which accesses only 20% tuples of the entire dataset, with the projection query passed in as a blackbox function. To compare with the visualization on the full result, we show false positives and false negatives of $\tilde{\mathcal{S}}_{\text{sparse}}$ w.r.t. $\mathcal{S}_{\text{sparse}}$ as grey and black dots, respectively. It is obvious in both Figure 6.1a and 6.1b that Love's 31-point, 31-rebound performance was impressive, being far from the vast majority of all points. One can also see the resemblance between the two figures, even though Figure 6.1b incurs significantly fewer data accesses and hence much lower latency.

[4] We set $r_x = r_y = 2$ and $\tau = 16$ such that $\mathcal{S}_{\text{sparse}}$ contains roughly 100 points so that human eye can perceive.

6.2.3 Query Types

We study the following three types of exploration queries commonly made on sports data. In Section 6.6.1, we also show applications to two other domains, the *Computer Science bibliography* and the *Wikipedia edit history*.

I. **Projection Query.** Given numerical attributes $A, B \in R$,

$$f(\mathcal{R}) = \{(t.A, t.B) \mid t \in \mathcal{R}\}.$$

An example application of the projection query would be to find a player's point-rebound stats in every game; i.e., $(A, B) = (\texttt{points}, \texttt{rebounds})$. When f is applied to Kevin Love, one of the result 2d points would be $(31, 31)$, which corresponds to "Kevin Love's 31-point, 31-rebound game."

II. **Count Query.** Given numerical attribute $A \in R$,

$$f(\mathcal{R}) = \{(v, c) \mid v \in \mathcal{R}.A \wedge c = |\{t \in \mathcal{R} \mid t.A \geqslant v\}|\}.$$

For example, Michael Jordan's 38 games with 50 or more points (most in NBA history[5]) maps to a 2d point $(50, 38)$ when evaluating a count query on Jordan's game-by-game stats data with $A = \texttt{points}$. Note that f can return multiple points for Michael Jordan, with lower point values associated with higher count values.

III. **Streak Query.** Given numerical attribute $A \in R$,

$$f(\mathcal{R}) = \text{Pareto optimal subset of } \{(v, \mathsf{len} = r - l + 1) \mid$$
$$1 \leqslant l \leqslant r \leqslant n_{\mathcal{R}} \ \wedge \ v = \min_{l \leqslant i \leqslant r} t_i.A\}.$$

[5] `www.nba.com/jordan/list_50games.html`

This is known as the set of *prominent streaks* in sequence $\mathcal{R}.A$ Jiang et al. (2011). For instance, LeBron James's 9-game 35-or-more-points streak is represented by a 2d point $(35, 9)$ in the result set of evaluating a streak query on James' stats (with $A = \texttt{points}$).

While these three types of exploration queries are represented by very different functions f, they share one common characteristic—same formats of input (a relation conforming to schema R) and output (a set of points in $\mathcal{R}$). In the rest of this chapter, we illustrate our algorithms using these query types.

6.3 System Overview

In this section, we first describe our storage structure for data, and then walk through the flow of the algorithm.

6.3.1 Data Storage and Working Memory

From a system point of view, we would like to host the exploration query evaluation service for multiple datasets. Hence, we will not dedicate memory to storing the entire dataset, even if it fits into memory by itself. Instead, the dataset is stored in SSTable (Chang et al., 2008), a distributed key-value storage system supporting efficient lookup. The data of an object is accessed via a service API using its index as the key; additionally, *any single tuple* can be accessed using the object index along with the tuple index as the key. In other words, the API allows both object- and tuple-level access.

On the other hand, we do assume that a small amount of memory is reserved for storing a small sample of the dataset, and that enough working memory is available, on a per-query basis, for storing data accessed during query evaluation (whose size is capped by budget η) and for query evaluation.

6.3.2 Workflow

Here we describe the high-level workflow of evaluating an exploration query (Figure 6.2). Detailed algorithms and analysis will be provided in Sections 6.4 and 6.5, respectively.

0. Upon initialization of the query evaluator, we establish the connection to the dataset $\mathcal{D}$, and "prefetch" a random sample of $\zeta|\mathcal{D}|$ tuples into memory, where ζ is the sample rate and $|\mathcal{D}|$ is the total number of tuples in $\mathcal{D}$. We leave the the details of this prefetching step to Section 6.4.

   Since the prefetching step is performed only once and its cost is amortized over all ensuing exploration queries regardless of their types, we do not count its data accesses towards the budget η in our problem definition. On the other hand, the sample rate ζ is bounded by the amount of memory reserved for storing the sample, which is far less than the total data size.

1. When an exploration query comes in, f is passed in as a blackbox function, with attribute(s) specified as parameter(s) of f.

2. At its discretion, the algorithm executes function f on the prefetched sample, and/or full data for a subset of the objects retrieved via the object-level API. Depending on the algorithm, this step may be performed more than once.

3. Based on the results of executing f in the previous step, the algorithm computes the set $\mathcal{S}_{\text{sparse}}$ of sparse points and a sketch of the remaining points $\mathcal{S}\backslash\mathcal{S}_{\text{sparse}}$, or approximations thereof.

4. The results are then combined to produce a visualization for the result of the exploration query, using a scatter plot overlaid on top of a heatmap.

Algorithm 11: ExploreBase(f, r_x, r_y, τ)

```
1  S ← ∅;
2  for i = 1 to N do
3  |   R_i ← LoadObjectData(i);
4  |   S ← S ⊎ f(R_i);
5  S_sparse ← ∅; S_sketch ← ∅;
6  foreach p ∈ S do
7  |   if |N_S(p; r_x, r_y)| ⩽ τ then
8  |   |   S_sparse ← S_sparse ⊎ {p};
9  |   else
10 |   |   S_sketch ← S_sketch ⊎ {(p, 1)};
11 return S_sparse, S_sketch;
```

In this workflow, the evaluator assumes that it knows only the input and output format of f, and nothing about how f actually processes the input and produces the output. In other words, an exploration query can be represented by any blackbox function f carrying the signature specified in Section 6.2.1, with a few examples given in Section 6.2.3.

6.4 Algorithms

In this section, we present two algorithms. The baseline algorithm (Algorithm 11) performs exact evaluation of an exploration query f, ignoring computation budget η, and computes accurate ($\mathcal{S}_{\text{sparse}}$, $\mathcal{S}_{\text{sketch}}$) for the full result set $\mathcal{S}$. The sampling-based algorithm (Algorithm 12) produces approximate solutions to $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ more efficiently than the baseline, within the computation budget η.

6.4.1 Baseline Algorithm

We first present a straightforward algorithm (Algorithm 11) that performs exact evaluation on the full dataset given an exploration query f. The algorithm takes as input (i) a callable function f specifying the exploration query, and (ii) radii r_x and

r_y that define the neighborhood of a point, and sparsity threshold τ.

Algorithm 11 evaluates the exploration query in a brute-force fashion. For each object i, we load its data $\mathcal{R}_i$ into memory (line 3). (If a tuple is in the prefetched sample, we can avoid reloading it; this detail is not shown in Algorithm 11.) Then, we feed object i's data as input to function f to evaluate (line 4). $\mathcal{S}$ is simply the union of execution output over all objects. For each point $p \in \mathcal{S}$, we include it in $\mathcal{S}_{\text{sparse}}$ if it has no more than τ neighbors in $\mathcal{S}$. Otherwise, we simply create a sketch point for it with weight 1 in $\mathcal{S}_{\text{sketch}}$.

Counting Neighbors. Neighbor counting by brute force can take as much as $O(|\mathcal{S}|^2)$ time. We use the following heuristic by partitioning the result set $\mathcal{S}$ using grid cells each of size $r_x \times r_y$.

By Lemma 8, the neighborhood of a point $p \in \mathcal{R}$ contains the grid cell $\square_{ij}$ containing p, and is contained by the 3×3 grid cells centered at $\square_{ij}$. Moving from $\mathcal{R}$ to the finite result set $\mathcal{S}$, for a $p \in \mathcal{S}$ that belongs to some partition $\mathcal{P}_{ij} = \mathcal{S} \cap \square_{ij}$, p's neighbors in $\mathcal{S}$, i.e., $\mathcal{N}_{\mathcal{S}}(p; r_x, r_y)$, form a superset of $\mathcal{P}_{ij}$ and fall within the 3×3 partitions centered at $\mathcal{P}_{ij}$ (Corollary 9).

Lemma 8. *For any $p \in \mathcal{R}$, if $p \in \square_{ij}$, where $\square_{ij} = [i \cdot r_x, (i+1) \cdot r_x) \times [j \cdot r_y, (j+1) \cdot r_y)$, then we have*

i. $\square_{ij} \subseteq \mathcal{N}_{\mathcal{R}}(p; r_x, r_y)$;

ii. $\mathcal{N}_{\mathcal{R}}(p; r_x, r_y) \subseteq \bigcup_{i'=i-1}^{i+1} \bigcup_{j'=j-1}^{j+1} \square_{i'j'}$.

Corollary 9. *Given $\mathcal{S} \subseteq \mathcal{R}$, partition it as $\mathcal{S} = \bigcup_{i,j \in \mathbb{Z}} \mathcal{P}_{ij}$, where $\mathcal{P}_{ij} = \mathcal{S} \cap \square_{ij}$. For any $p \in \mathcal{P}_{ij}$, we have*

i. $\mathcal{P}_{ij} \subseteq \mathcal{N}_{\mathcal{S}}(p; r_x, r_y)$;

ii. $\mathcal{N}_{\mathcal{S}}(p; r_x, r_y) \subseteq \bigcup_{i'=i-1}^{i+1} \bigcup_{j'=j-1}^{j+1} \mathcal{P}_{i'j'}$.

Because $\mathcal{S}$ is determined by f, whose behavior is assumed to be unknown to the algorithm, it is not possible to index $\mathcal{S}$ beforehand in order to speed up neighbor counting. However, with Corollary 9, we can avoid performing $O(|\mathcal{S}|^2)$ comparisons, and narrow down the possible neighbors of a point to its 9 adjacent partitions. Moreover, if a partition $\mathcal{P}_{ij}$ contains more than τ points, we can immediately determine that all its points should be added to the sketch.

Time Complexity. The execution of Algorithm 11 consists of three steps: (i) loading each object's data into memory, (ii) executing f on each object's data, and (iii) computing $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ from $\mathcal{S}$.

Step (i) requires accessing the full dataset, which requires fetching $(1-\zeta)\|\mathcal{D}\|$ tuples, with prefetched ones excluded. This use of the prefetched sample is not very effective; we will see a much better use of this sample in Section 6.4.2.

Steps (ii) and (iii) are carried out in memory. Step (ii) depends on how f behaves, while step (iii) only depends on the size of the result set $\mathcal{S}$ and the sparsity threshold τ.

For step (ii), the time complexity for the brute-force execution of f is linear in the number of tuples for all three types of exploration queries described in Section 6.2.3. Hence, the overall complexity of executing such queries on $\mathcal{D}$ is $O(\|\mathcal{D}\|)$.

For step (iii), thanks to the counting technique we have described, the worse case complexity is improved from $O(|\mathcal{S}|^2)$ to $O(\tau \cdot |\mathcal{S}|)$. The worse-case scenario is that each (non-empty) partition of $\mathcal{S}$ contains τ points (so no point can be pruned for counting), and each non-empty partition is adjacent to one or more (at most 8) other non-empty partition(s). In this case, the total number of pairs of points compared is $O\left(\tau^2 \cdot \frac{|\mathcal{S}|}{\tau}\right) = O(\tau \cdot |\mathcal{S}|)$. In particular, for all three types of exploration queries we consider, $|\mathcal{S}| = O(\|\mathcal{D}\|)$, i.e., linear in the size of the full dataset $\mathcal{D}$.

Result Quality. In terms of the quality of the output, without imposing the computation budget η, Algorithm 11 trivially returns the exact $\mathcal{S}_{\text{sparse}}$ and a sketch that gives $\delta(\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}, \mathcal{S}_{\text{sketch}}) = 0$.

6.4.2 Sampling-based Algorithms

The baseline algorithm essentially fetches the full dataset $\mathcal{D}$, object by object. Due to the large volume of the data, accessing the entire dataset piece by piece via service API can be very costly. The complexity of full evaluation and counting makes it inefficient to perform such brute-force evaluation in an interactive environment.

The baseline algorithm also disregards the computation budget η. Working under a budget constraint is challenging. Because we do not assume anything about the behavior of f, there is no guarantee that evaluating f on partial data of an object will produce a partial output of evaluating f on the object's full data. Therefore, in order to comply with the budget constraint, we must choose to evaluate f on the full data for a subset of objects, and completely ignore the remaining objects. But how can we know which subset of objects to choose, without knowing how f behaves? The challenge is compounded by the problem that the data $\mathcal{R}_i$ of each object i follows some unknown distribution. For example, in the context of the basketball data, an object could be a player or a team. For a player object, depending on the players' position on the court (e.g., point guard vs. power forward), and depending on the ability of the player (e.g., a superstar like Michael Jordan vs. a mediocre player), the distribution of this object's data will obviously be very different from other objects.

Fortunately, the prefetched sample comes to rescue. Suppose that for each object, we have a sample of its data. Then, the result of applying f on the prefetched sample may resemble the full result $\mathcal{S}$ in terms of outlier identities, even though the outliers returned by f, e.g., a *streak query*, represent low-probability events that come from one or a few tuples of the full data. In Section 6.5.1, we provide an analysis of this

Algorithm 12: ExploreSample$(f, \{\mathcal{R}_i^{\mathrm{S}}\}_{i=1}^{N}, r_x, r_y, \tau, \eta)$

$\mathcal{S}^{\mathrm{S}} \leftarrow \varnothing$;
for $i = 1$ **to** N **do**
  $\mathcal{S}^{\mathrm{S}} \leftarrow \mathcal{S}^{\mathrm{S}} \uplus f(\mathcal{R}_i^{\mathrm{S}})$;
$(\mathcal{O}^+, \mathcal{O}^-) \leftarrow$ SelectObjects$(\mathcal{S}^{\mathrm{S}}, \eta)$;
$\mathcal{S}^+ \leftarrow \varnothing$;
foreach $i \in \mathcal{O}^+$ **do**
  $\mathcal{R}_i \leftarrow$ LoadObjectData(i);
  $\mathcal{S}^+ \leftarrow \mathcal{S}^+ \uplus f(\mathcal{R}_i)$;
$\mathcal{S}^- \leftarrow \varnothing$;
foreach $j \in \mathcal{O}^-$ **do**
  $\mathcal{R}_j \leftarrow$ LoadObjectData(j);
  $\mathcal{S}^- \leftarrow \mathcal{S}^- \uplus f(\mathcal{R}_j)$;
$\tilde{\mathcal{S}}_{\mathrm{sparse}} \leftarrow \varnothing$; $\mathcal{S}_{\mathrm{sketch}} \leftarrow \varnothing$;
foreach $p \in \mathcal{S}^+$ **do**
  if $|\mathcal{N}_{\mathcal{S}^+}(p; r_x, r_y)| + \lambda \cdot |\mathcal{N}_{\mathcal{S}^-}(p; r_x, r_y)| \leqslant \tau$ **then**
    $\tilde{\mathcal{S}}_{\mathrm{sparse}} \leftarrow \tilde{\mathcal{S}}_{\mathrm{sparse}} \uplus \{p\}$;
  else
    $\mathcal{S}_{\mathrm{sketch}} \leftarrow \mathcal{S}_{\mathrm{sketch}} \uplus \{p, 1\}$;
foreach $p \in \mathcal{S}^-$ **do**
  $\mathcal{S}_{\mathrm{sketch}} \leftarrow \mathcal{S}_{\mathrm{sketch}} \uplus \{p, \lambda\}$;
return $\tilde{\mathcal{S}}_{\mathrm{sparse}}$, $\mathcal{S}_{\mathrm{sketch}}$;

connection between the result on the sample and that on the full data for *projection queries*.

In the remainder of this section, we describe a general sampling-based algorithm that uses the prefetched sample to select a small number of objects, for which we access their full data for evaluation (Algorithm 12).

Prefetching. Algorithm 12 assumes a prefetching step (Step 0 described in Section 6.3.2) that works as follows. Upon initialization of the query evaluator, given a fixed sample rate ζ, for each object i, we sample ζn_i times uniformly and independently at random from $t_1, \ldots, t_{n_i}$ with replacement. All the ζn_i sample tuples

form the sample data $\mathcal{R}_i^{\mathrm{S}}$ of object i. The full set of sample data $\{\mathcal{R}_i^{\mathrm{S}}\}_{i=1}^{N}$ sits in the memory throughout the lifetime of the evaluator, and is fed into the evaluation of each forthcoming exploration query f in Algorithm 12.

Objects Selection. Algorithm 12 first executes f on the prefetched sample for all objects. The result, denoted by $\mathcal{S}^{\mathrm{S}}$, is the union of $f(\mathcal{S}_i^{\mathrm{S}})$ along with the ownership of each point in the result (line 3). Base on $\mathcal{S}^{\mathrm{S}}$, we select two disjoint subsets of objects (line 4): (i) $\mathcal{O}^+$, the set of objects such that we envision for each $p \in \mathcal{S}_{\mathrm{sparse}}$, the object i that yields p under f (i.e., $p \in f(\mathcal{R}_i)$) belongs to $\mathcal{O}^+$, and (ii) $\mathcal{O}^-$, a random sample of all objects $\mathcal{O}$ excluding $\mathcal{O}^+$ (i.e., $\mathcal{O} \backslash \mathcal{O}^+$). We ensure that the total number of tuples contained in the full data for these objects is within the budget η. We defer the discussion of how the objects are selected (line 4) to Section 6.5.

Full Execution. The two sets of objects $\mathcal{O}^+$ and $\mathcal{O}^-$ are presumably much smaller than the set of all objects. Algorithm 12 then executes f on the full data for $\mathcal{O}^+$ and $\mathcal{O}^-$; we denote the result sets as $\mathcal{S}^+$ and $\mathcal{S}^-$ respectively.

Counting. There are two key differences in neighbor counting and result generation compared to Algorithm 11: (1) only points of $\mathcal{S}^+$ may be included in $\tilde{\mathcal{S}}_{\mathrm{sparse}}$, and (2) while each point of $\mathcal{S}^+$ is counted once as before, the presence of each point in $\mathcal{S}^-$ is multiplied by λ, a multiplier depending on how the budget η is divided between $\mathcal{O}^+$ and $\mathcal{O}^-$. The choice of the value of λ will also be described in Section 6.5.

Time Complexity. Compared to the complexity of the Algorithm 11, Algorithm 12 reduces the cost of fetching data and counting from $O(\|\mathcal{D}\|)$ to $O((\zeta + \eta) \cdot \|\mathcal{D}\|)$. For cost of execution f for a query with linear/super-linear complexity $T(n)$, the overall time complexity is reduced to $T((\zeta + \eta) \cdot \|\mathcal{D}\|)$. Specifically, for the three types of

queries we consider with linear time complexity, the overall time complexity is also reduced to $O((\zeta + \eta) \cdot \|\mathcal{D}\|)$.

Difficulty of Provisioning $\mathcal{O}^+$. It is obvious that the choice of $\mathcal{O}^+$ is critical to the quality of $\tilde{\mathcal{S}}_{\text{sparse}}$—if $i \notin \mathcal{O}^+$, there is no chance for points of $f(\mathcal{R}_i)$ to appear in $\tilde{\mathcal{S}}_{\text{sparse}}$. To understand why it is necessary to perform online selection of $\mathcal{O}^+$ and $\mathcal{O}^-$ based on $\mathcal{S}^{\text{S}}$, one must consider the diversity of queries that can be applied to a single dataset.

Figure 6.3 shows the scatter plot with heatmap of accurate results for two queries of the same type (projection) but on different attributes. Query f_1 projects the NBA players' game-by-game performance to the `points-rebounds` plane, while f_2 projects the same data on two different attributes `assists` and `steals`. For each of f_1 and f_2, we use a rectangular neighborhood of size approximately $\frac{1}{10} \times \frac{1}{10}$ of the result space, and a proper value of τ that limits the size of $\mathcal{S}_{\text{sparse}}$ to be roughly 100. Comparing $\mathcal{S}^1_{\text{sparse}}$ for f_1 and $\mathcal{S}^2_{\text{sparse}}$ for f_2, $\mathcal{S}^1_{\text{sparse}}$ has 99 points (possibly overlapping) corresponding to 40 distinct objects, and $\mathcal{S}^2_{\text{sparse}}$ has 101 points corresponding to 46 distinct objects. Together, $\mathcal{S}^1_{\text{sparse}}$ and $\mathcal{S}^2_{\text{sparse}}$ consist of a total of 83 distinct objects, sharing only 3 in common.

This example illustrates that it is impossible to use a static choice of $\mathcal{O}^+$ to provide a good coverage of objects that result in points of $\mathcal{S}_{\text{sparse}}$ for all possible queries. Therefore, we perform query-specific online object selection, as we explain in the next section.

6.5 Objects Selection

In this section, we discuss how $\mathcal{O}^+$ and $\mathcal{O}^-$ are selected in Algorithm 12 (line 4).

At a high level, this process is analogous to the problem of computing a small *coreset* of a large point set that can be used for approximating various "extent

measures" of the full point set (Agarwal et al., 2005). More specifically, for the exploration query evaluation problem, we would like to approximate two types of extent measures—the sparse points $\mathcal{S}_{\text{sparse}} \subseteq \mathcal{S}$, and a density estimate for all other points $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$. The challenge here is that we do not have direct access to the full point set $\mathcal{S}$. For example, as we treat the evaluation function f as a blackbox, there is no way to sample points uniformly and independently from $\mathcal{S}$ without computing it first, i.e., performing full evaluation on all objects. In other words, with an f whose behavior is unknown to Algorithm 12, the coreset we construct cannot be an arbitrary subset of $\mathcal{S}$. It has to be the result point set obtained by evaluating f for a subset of the objects in $\mathcal{O}$.

Overall, given a budget η on the number of tuples in $\{\mathcal{R}_i\}_{i \in \mathcal{O}^+ \cup \mathcal{O}^-}$, we first select $\mathcal{O}^+$ with budget η^+, and then use whatever is left of the budget to select $\mathcal{O}^-$.

6.5.1 Selecting $\mathcal{O}^+$

The objects selection step in Algorithm 12 (especially the choice of $\mathcal{O}^+$) is crucial to the efficiency of the algorithm and to the quality of approximation ($\tilde{\mathcal{S}}_{\text{sparse}}$, $\mathcal{S}_{\text{sketch}}$) to the actual solution ($\mathcal{S}_{\text{sparse}}$, $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$). A large $\mathcal{O}^+$ will give $\tilde{\mathcal{S}}_{\text{sparse}}$ a good coverage of $\mathcal{S}_{\text{sparse}}$. However, a large $\mathcal{O}^+$ also leads to slower evaluation. Given a fixed budget η^+ on the number of tuples allowed to be accessed, efficiency would not be an issue anymore, as the number of tuples to be evaluated is limited, so the question becomes how to choose $\mathcal{O}^+$ of limited size in order to maximize the recall of $\tilde{\mathcal{S}}_{\text{sparse}}$ with respect to $\mathcal{S}_{\text{sparse}}$.

Observe that for any point $p \in \mathcal{S}_{\text{sparse}}$ where $p \in f(\mathcal{R}_i)$ (i.e., p comes from object i), it is a necessary condition that $i \in \mathcal{O}^+$ for p to possibly appear in $\tilde{\mathcal{S}}_{\text{sparse}}$. We assume that if an object produces outlier points of $\mathcal{S}$ (when f is evaluated on the full data), it is likely to also produce outlier points when the same query f is evaluated on a sample of its data. Therefore, the prefetched sample serves as a good guide for

choosing $\mathcal{O}^+$. We take a partition-based approach to address this problem.

Recall that $\mathcal{S}^{\mathrm{S}}$ denotes the result of executing f on the prefetched sample for all objects (Algorithm 12, line 3). To select objects for $\mathcal{O}^+$, we first partition $\mathcal{S}^{\mathrm{S}}$ into $\mathcal{P}^{\mathrm{S}}$ into grid cells each of size $r_x^{\mathrm{S}} \times r_y^{\mathrm{S}}$. More precisely, $\mathcal{P}^{\mathrm{S}} = \bigcup_{i,j \in \mathbb{Z}} \mathcal{P}_{ij}^{\mathrm{S}}$, where $\mathcal{P}_{ij}^{\mathrm{S}} = \mathcal{S}^{\mathrm{S}} \cap \square_{ij}^{\mathrm{S}}$, $\square_{ij}^{\mathrm{S}} = [i \cdot r_x^{\mathrm{S}}, (i+1) \cdot r_x^{\mathrm{S}}) \times [j \cdot r_y^{\mathrm{S}}, (j+1) \cdot r_y^{\mathrm{S}})$. Based on the partitioning $\mathcal{P}^{\mathrm{S}}$ and the point-object ownership relation, we propose two strategies, namely sparsest-grid-cell (Section 6.5.1) and sparsest-object (Section 6.5.1) for choosing O^+ objects, given an upper bound η^+ on the data size (in number of tuples).

Sparsest Grid Cell

The sparest-grid-cell strategy works as follows. Examine the partitions in non-descending order of their sizes. For a partition $\mathcal{P}_{ij}^{\mathrm{S}}$, include in $\mathcal{O}^+$ all objects that contribute at least one point in $\mathcal{P}_{ij}^{\mathrm{S}}$, i.e., $\mathcal{O}^+ = \mathcal{O}^+ \cup \{k \mid \mathcal{S}_k^{\mathrm{S}} \cap \mathcal{P}_{ij}^{\mathrm{S}} \neq \varnothing\}$. Terminate when the budget η^+ is reached.

We illustrate the idea behind this strategy using the *projection query*. The projection query simply projects the high-dimensional tuples onto a plane defined by two given attributes. Thus, the result on sample $\mathcal{S}^S$ is a random sample of the full result $\mathcal{S}$ of size $\zeta \cdot \|\mathcal{D}\|$.

It is known (Vapnik and Chervonenkis, 1971; Phillips, 2012) that for a d-dimensional point set P of size n, and a random sample S of P of size $k = (d/\epsilon^2)\log(2n/\delta)$, with probability at least $1 - \delta$, for all d-dimensional axis-aligned rectangle R:

$$\left| \frac{|P \cap R|}{|P|} - \frac{|S \cap R|}{|S|} \right| \leqslant \epsilon.$$

For projection query, let $P = \mathcal{S}$, $k = \zeta \cdot \|\mathcal{D}\|$, $r_x^{\mathrm{S}} = r_x$, and $r_y^{\mathrm{S}} = r_y$, it follows that with probability at least $1 - 2 \cdot \|\mathcal{D}\| \cdot \exp(\zeta \cdot \|\mathcal{D}\| \epsilon^2/2)$, for all $q \in \mathcal{S}^{\mathrm{S}}$:

$$\left| \frac{|\mathcal{N}_{\mathcal{S}}(q; r_x, r_y)|}{\|\mathcal{D}\|} - \frac{|\mathcal{N}_{\mathcal{S}^{\mathrm{S}}}(q; r_x^{\mathrm{S}}, r_y^{\mathrm{S}})|}{\zeta \cdot \|\mathcal{D}\|} \right| \leqslant \epsilon.$$

This means, with high probability, for all points of $\mathcal{S}^{\mathrm{S}}$, its neighborhood density is close to its neighborhood density in $\mathcal{S}$. Therefore, a point of $\mathcal{S}^{\mathrm{S}}$ with a sparser neighborhood has a higher probability of being present in $\mathcal{S}_{\text{sparse}}$.

Similar analysis can be conducted for the other query types as well. While the sparsest-grid-cell strategy is oblivious to the behavior of the query evaluation function f, it follows the idea behind the analysis above by choosing points with sparest neighborhood.

By Corollary 9, for any point p in a partition $\mathcal{P}_{ij}^{\mathrm{S}}$, its number of neighbors $|\mathcal{N}_{\mathcal{S}^{\mathrm{S}}}(p; r_x^{\mathrm{S}}, r_y^{\mathrm{S}})|$ is bounded from below by $|\mathcal{P}_{ij}^{\mathrm{S}}|$. We use this lower bound instead of counting the exact number of neighbors for each point of $\mathcal{S}^{\mathrm{S}}$ for efficiency reasons.

Sparsest Object

An object that contributes to $\mathcal{S}_{\text{sparse}}$ might not be selected by the sparsest-grid-cell strategy due to an "unfortunate" draw of sample. However, if the "overall quality" of the object is good, we can hope to reduce the role of luck in this process by considering the overall sparsity of points produced by this object in $\mathcal{S}^{\mathrm{S}}$.

For each object i, we define its overall sparsity as

$$\mu_i = \mathsf{mean}_{p \in \mathcal{S}^{\mathrm{S}} \cap f(\mathcal{R}_i^{\mathrm{S}})} \{ |\mathcal{P}_{ij}^{\mathrm{S}}| \mid p \in \mathcal{P}_{ij}^{\mathrm{S}} \}.$$

In other words, for each object i, we consider the mean neighborhood sparsity of all points in $\mathcal{S}^{\mathrm{S}}$ that are produced by object i. Again, the partition size is used as an approximation for the actual number of neighbors for efficiency reasons.

In fact, the sparsest-grid-cell strategy can be considered as a special case of the sparsest-object strategy with $p = -\infty$ for the power mean function

$$\mathcal{M}_p(x_1, x_2, ...x_n) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^p\right)^{1/p} \tag{6.5}$$

It is known that $\mathcal{M}_{-\infty}(x_1, x_2, ..., x_n) = \min(x_1, x_2, ..., x_n)$, representing the strategy deployed by sparsest-grid-cell.

We experiment this strategy with three other instantiations of the power mean function $\mathcal{M}$, namely *arithmetic mean* ($p = 1$), *geometric mean* ($p = 0$), and *harmonic mean* ($p = -1$).

6.5.2 Selecting $\mathcal{O}^-$

We adopt a simple strategy for selecting the object set $\mathcal{O}^-$—given $\mathcal{O}^+$, include each object of $\mathcal{O}\backslash\mathcal{O}^+$ independently with probability p. The multiplier in Algorithm 12 is set to $\lambda = 1/p$ to maintain expectation, i.e. for any point in $\mathcal{S}\backslash\mathcal{S}^+$, its expected frequency in $\mathcal{S}^-$ is $\lambda \cdot p = 1$.

Another way to think of this strategy is to sample points from $\mathcal{S}$ with correlation. For any object $i \in \mathcal{O}\backslash\mathcal{O}^+$, the sample point set $\mathcal{S}^-$ either includes all points of $f(\mathcal{R}_i)$, or none of it.

$\mathcal{O}^-$ affects the quality of output by Algorithm 12 in several ways. First, not all points of $\mathcal{S}^+$ lead to points in $\mathcal{S}_{\text{sparse}}$ after evaluating f on $\mathcal{O}^+$. It is up to $\mathcal{O}^-$ and $\mathcal{S}^-$ to exclude false positives and include true positives (line 15). Second, the quality of $\mathcal{S}_{\text{sketch}}$ is determined primarily by $\mathcal{S}^-$ (line 20).

Budget Constraint

Note that we need to comply with the total budget constraint η. Let $X_i \sim Ber(p)$ be the Bernoulli random variable denoting if object i is chosen to be included in $\mathcal{O}^-$, for $i \in \mathcal{O}\backslash\mathcal{O}^+$. All X_i's are independent of each other. Let Y be the total number of tuples for objects in $\mathcal{O}^-$. Let $\|\mathcal{O}^+\|$ denote the number of tuples for objects in

$\mathcal{O}^+$. Following notations from Section 6.2.1, the expected number of tuples for $\mathcal{O}^-$ is given by

$$\mu_Y = \mathbb{E}\left[\sum_{i\in\mathcal{O}\setminus\mathcal{O}^+} X_i n_i\right] = \sum_{i\in\mathcal{O}\setminus\mathcal{O}^+} \mathbb{E}[X_i] n_i = p\cdot(\|\mathcal{D}\| - \|\mathcal{O}^+\|).$$

The variance in the number of tuples in $\mathcal{O}^-$ is given by

$$\sigma_Y^2 = \mathrm{Var}\left[\sum_{i\in\mathcal{O}\setminus\mathcal{O}^+} X_i n_i\right] = p(1-p)\cdot\sum_{i\in\mathcal{O}\setminus\mathcal{O}^+} n_i^2.$$

By (one-sided) Chebyshev's inequality, we have

$$\mathbf{Pr}\left[Y \geqslant (1+\Delta)\cdot\mu_Y\right] \leqslant \frac{1}{1+(\Delta\mu_Y/\sigma_Y)^2}.$$

By setting $(1+\Delta)\mu_Y = \eta\cdot\|\mathcal{D}\| - \|\mathcal{O}^+\|$, we have

$$\mathbf{Pr}\left[Y + \|\mathcal{O}^+\| \geqslant \eta\cdot\|\mathcal{D}\|\right] \leqslant \frac{1}{1+(\Delta\mu_Y/\sigma_Y)^2},$$

where

$$\Delta = \frac{\eta\cdot\|\mathcal{D}\| - \|\mathcal{O}^+\|}{\mu_Y} - 1.$$

Since μ_Y is monotone in p, choosing a smaller value of p gives a better chance of complying with the budget constraint η.

Quality of $\tilde{\mathcal{S}}_{sparse}$

We study how $\mathcal{O}^-$ affects the quality of $\tilde{\mathcal{S}}_{\text{sparse}}$ in two ways.

1. For a point in $\mathcal{S}^+ \cap \mathcal{S}_{\text{sparse}}$, what is the minimum probability that it is included in $\tilde{\mathcal{S}}_{\text{sparse}}$?

2. For a point in $\mathcal{S}^+ \backslash \mathcal{S}_{\text{sparse}}$, what is the maximum probability that it is included in $\tilde{\mathcal{S}}_{\text{sparse}}$?

For a point $q \in \mathcal{S}^+ \cap f(\mathcal{R}_i)$, i.e., a point of $\mathcal{S}^+$ coming from object i, let $C_j = |\mathcal{N}_{\mathcal{S}}(q) \cap f(\mathcal{R}_j)|$ be the number of neighbor of q in $\mathcal{S}$ coming from object $j \neq i$. We attempt to provide a bound for the two questions above in the worst case, where all q have at least one neighbor in $\mathcal{S}^+$. Let random variable Z denote the estimated number of q's neighbors. Following the notation of random variable X_i's from Section 6.5.2, Z can be written as follows:

$$Z = 1 + \lambda \cdot \sum_{j \neq i} X_j C_j.$$

And we have

$$\mu_Z = 1 + \sum_{j \neq i} C_j = |\mathcal{N}_{\mathcal{S}}(q)|, \quad \sigma_Z^2 = \frac{1-p}{p} \cdot \sum_{j \neq i} C_j^2.$$

1. If $\mu_Z \leqslant \tau$, by Chebyshev's inequality, we have

$$\mathbf{Pr}[Z > \tau] \leqslant \frac{1}{1 + (\mu_Z - \tau)^2 / \sigma_Z^2}.$$

2. If $\mu_Z > \tau$, symmetrically, we have

$$\mathbf{Pr}[Z \leqslant \tau] \leqslant \frac{1}{1 + (\mu_Z - \tau)^2 / \sigma_Z^2}.$$

These bounds suggest a couple of things. First, the farther μ_Z deviates from τ, the more confident we can be that Algorithm 12 will make the right decision on whether to include q in $\tilde{\mathcal{S}}_{\text{sparse}}$. In other words, it is harder to classify points correctly whose actual neighborhood density in $\mathcal{S}$ is close to the sparsity threshold τ. Also, since τ is presumably small, in general it is harder to classify points of $\mathcal{S}_{\text{sparse}}$ correctly than $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$, i.e., high recall is harder to achieve than high precision. Second, a larger budget for $\mathcal{O}^-$ would lead to a larger p, thus higher confidence in classification.

Quality of $\mathcal{S}_{sketch}$

Recall that the sketch distance (defined in Section 6.2.2) is used to measure the quality of a sketch $\mathcal{S}_{\text{sketch}}$ w.r.t. $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$. While the exact sketch distance is not easy to come by, an upper bound can be obtained as follows.

Ignore $\mathcal{S}_{\text{sparse}}$ for now. Partition the result set $\mathcal{S}$ such that any two points in the same partition are neighbors. For example, under L_∞-norm neighborhood definition, partition $\mathcal{S}$ into grid cells each of size $r_x \times r_y$. Let $P_1, \ldots, P_m$ denote the resulting partitions. Let $Z_j = \lambda \cdot |\mathcal{S} \cap P_j|$ be the estimated number of points in partition P_j by the sketch. Let $Z = \sum_j Z_j$ be the estimated total number of points.

We have the following bound on $\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}})$, expressed in terms of the means and variances of Z_j's and Z.

$$\mathbf{Pr}\left[\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}}) \geqslant 1 - \frac{1-\Delta^-}{1+\Delta^+}\right] \leqslant \sum_j \frac{1}{1 + (\Delta^- \mu_j/\sigma_j)^2} + \frac{1}{1 + (\Delta^+ \mu_Z/\sigma_Z)^2}. \quad (6.6)$$

Tighter bounds can be obtained by taking into account the correlation among Z_j's. When the distributions of number of points by objects are identical in all partitions, we have

$$\mathbf{Pr}\left[\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}}) \geqslant 1 - \frac{1-\Delta}{1+\Delta}\right] \leqslant \frac{2}{1 + (\Delta \mu_Z/\sigma_Z)^2}. \quad (6.7)$$

If, on top of the identical distribution scenario, C_i's are all equal, we have $\frac{\mu_Z^2}{\sigma_Z^2} = \frac{Np}{1-p}$, and

$$\mathbf{Pr}\left[\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}}) \geqslant 1 - \frac{1-\Delta}{1+\Delta}\right] \leqslant \frac{2}{1 + \Delta^2 Np/(1-p)}. \quad (6.8)$$

On the other hand, in the worse case where all Z_j's are independent and $\mu_j = \sigma_j$, we would have

$$\mathbf{Pr}\left[\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}}) \geqslant 1 - \frac{1-\Delta}{1+\Delta}\right] \leqslant \frac{N+1}{1+\Delta^2 p/(1-p)}. \tag{6.9}$$

Taking $\mathcal{S}_{\text{sparse}}$ back into account, since the frequency counts in $\mathcal{S}^+$ are all precise (estimates with zero variance), including $\mathcal{S}_{\text{sparse}}$ does not nullify any of the results above.

Proofs of the above bounds and additional remarks can be found in the appendix. Note that these bounds may still be quite loose. We will show the quality of sketch produced by Algorithm 12 via empirical results in Section 6.6.

6.6 Experiments

We implemented our algorithms in C++ and evaluated the efficiency and result quality on three real datasets. All experiments are conducted on a machine with Intel Core i7-2600 3.4GHz CPU and 7.8GB RAM.

6.6.1 Datasets

NBA players' game-by-game performance statistics (NBA)[6]: By considering a player as an object and his performance stats in each game as a tuple, the dataset consists of 1.01 million tuples ($\|\mathcal{D}\| \approx 1.01 \times 10^6$) for a total of 3,129 players ($n = 3129$).

DBLP[7]: For the DBLP data set, we consider authors as objects, and year-by-year performance of an author in terms of the author's numbers of publications in different venues; i.e., a tuple represents an author's performance in a given year,

[6] `http://www.basketball-reference.com`

[7] `http://dblp.uni-trier.de`

where attributes represent the author's publication counts in different venues in that year. This dataset consists of ∼67k tuples ($\|\mathcal{D}\| \approx 67 \times 10^3$) for ∼32k distinct authors ($n \approx 32 \times 10^3$)

Wikipedia edit history data (WIKI)[8]**:** The raw Wikipedia edit log in the main namespace consists of a total of ∼116.6 million entries, each representing a revision of an article, with information of the user ID, article category, edit size, etc. We consider users as objects, and a tuple is the (number, size) of all edits and minor edits in one day. For over 3.3 million users ($n \approx 3.3 \times 10^6$), we have a total of ∼8.91 million tuples ($\|\mathcal{D}\| \approx 8.91 \times 10^6$).

6.6.2 Quality Evaluation

We evaluated the quality of $(\tilde{\mathcal{S}}_{\text{sparse}}, \mathcal{S}_{\text{sketch}})$ in two aspects—(i) quality of $\tilde{\mathcal{S}}_{\text{sparse}}$ w.r.t. $\mathcal{S}_{\text{sparse}}$ in terms of recall and, less importantly, precision, and (ii) quality of $\mathcal{S}_{\text{sketch}}$ w.r.t. $\mathcal{S}\backslash\mathcal{S}_{\text{sketch}}$ as measured by the sketch distance function δ.

Note that the baseline algorithm (Algorithm 11) performs full evaluation, and hence produces the exact $\mathcal{S}_{\text{sparse}}$. Also, under L_∞-norm (rectangular neighborhood), the grid-partition-based baseline algorithm trivially achieves $\delta(\mathcal{S}_{\text{sketch}}, \mathcal{S}\backslash\mathcal{S}_{\text{sparse}}) = 0$. Therefore, we are only interested in evaluating the quality of results of the sampling-based algorithm (Algorithm 12).

The same set of experiments were conducted on all three datasets. We first illustrate observations common to all data sets using the NBA data set.

Impact of $\mathcal{O}^+$ selection strategy and neighborhood definition. We tested the performance of Algorithm 12 with (i) different strategies for selecting $\mathcal{O}^+$ described in Section 6.5.1 ($\mathcal{M}_p$ with $p = 1, 0, -1, -\infty$), and (ii) different $\|\cdot\|$ functions in the neighborhood definition (Eq. 6.2.1), namely L_1-, L_2-, and L_∞-norms, corresponding to diamon-

[8] `https://snap.stanford.edu/data/wiki-meta.html`

doid, oval, and rectangular neighborhood, respectively. No significant difference was observed in the quality of result produced by Algorithm 12, so figures are omitted.

Varying sample rate ζ. Performance of Algorithm 12 at different sample rates ζ is shown in Figure 6.4. Results suggest that increasing ζ gives notable improvement in the quality of $\tilde{\mathcal{S}}_{\text{sparse}}$ in the beginning (Figures 6.4a and 6.4b), but the improvement diminishes afterwards (beyond 5% for NBA). On the other hand, ζ has almost no impact on the quality of the sketch $\mathcal{S}_{\text{sketch}}$ in terms of distance to the exact solution $\mathcal{S} \backslash \mathcal{S}_{\text{sparse}}$ (Figure 6.4c).

Varying budget η^+ for $\mathcal{O}^+$. In Figure 6.5, we fix ζ and η^-, and observe the performance of Algorithm 12 with varying budget η^+. We see that as η^+ increases, precision and recall keep increasing and approach 100% (Figures 6.5a and 6.5b). Similar to the results for varying ζ, η^+ shows little impact on the quality of $\mathcal{S}_{\text{sketch}}$ (Figure 6.5c).

Varying budget η^- for $\mathcal{O}^-$. In contrast to ζ and η^+, increasing η^- gives big improvement on the quality of $\mathcal{S}_{\text{sketch}}$ (Figure 6.6c), while recall of $\tilde{\mathcal{S}}_{\text{sparse}}$ is barely affected (Figure 6.5b). It is worth noting that having too few objects $\mathcal{O}^-$ may increase the amount of false positives in $\tilde{\mathcal{S}}_{\text{sparse}}$, accounting for the increase in precision shown in Figure 6.5a.

Varying overall budget η. Results above show how η^+ or η^-, when the other is fixed, affects different aspects of result quality. Now we show the tradeoff between η^+ and η^- when fixing the overall budget η. In Figure 6.7, we fix the total budget $\eta = \eta^+ + \eta^- = 20\%$, and show the three quality indicators, varying the budget allocation between η^+ and η^-. From the earlier experiments, it is expected that a larger η^+ would lead to better precision and recall, while a larger η^- would lead to a better sketch with a smaller distance to the ground truth.

To measure the overall quality of a solution by Algorithm 12, we use a generalized version of the F-score, by taking the weighted harmonic mean of the three quality measure, precision, recall, and distance. We assign weights β_P^2, β_R^2, β_δ^2 to precision, recall, and $1-\delta$, respectively, where $\beta_R > \beta_\delta > \beta_P > 0$, signifying decreasing importance. Formally,

$$F_{\beta_P,\beta_R,\beta_\delta} = \frac{\beta_P^2 + \beta_R^2 + \beta_\delta^2}{\frac{\beta_P^2}{P} + \frac{\beta_R^2}{R} + \frac{\beta_\delta^2}{1-\delta(\mathcal{S}_{\text{sketch}}, \mathcal{S}\setminus\mathcal{S}_{\text{sparse}})}}. \tag{6.10}$$

We show results of two F-scores, $F_{1,3,2}$ and $F_{2,4,3}$. In Figure 6.7a and 6.7b, both $F_{1,3,2}$ and $F_{2,4,3}$ peak at $\eta^+ = 14\%$, leaving η^- at 6%. On the contrary, for streak query (Figure 6.7c), the quality of $\mathcal{S}_{\text{sketch}}$ is still decent even with η^- as small as 2%, which allows the majority of the overall budget to be allocated to $\mathcal{O}^+$ in order to improve the quality of $\tilde{\mathcal{S}}_{\text{sparse}}$.

Precision versus recall. From the results on NBA, we see a higher precision than recall when sufficient budget is given. The same holds for results on DBLP and WIKI. As we have shown in Section 6.5.2, $\mathcal{S}^-$ plays an important role in both eliminating false positives from and keeping true positives in $\mathcal{S}^+$. While the probability of false positives only depends on the rate at which objects are sampled from $\mathcal{O}\setminus\mathcal{O}^+$ to form $\mathcal{O}^-$, the probability of false negatives is conditional on the fact that the object that yields some point of $\mathcal{S}^+ \cap \tilde{\mathcal{S}}_{\text{sparse}}$ is included in $\mathcal{O}^+$ in the first place. And we discuss next why it is hard to ensure the right objects are chosen for $\mathcal{O}^+$.

We should note that the effectiveness of $\mathcal{O}^+$ selection depends on both the data distributions and the query type. Algorithm 12 assumes neither. To see how the data distribution affects the effectiveness of $\mathcal{O}^+$, we compare the recall of $\tilde{\mathcal{S}}_{\text{sparse}}$ on NBA and on WIKI. In Figure 6.8a, we show the recall of $\tilde{\mathcal{S}}_{\text{sparse}}$ on WIKI data with varying sample rate. The result is significantly worse than that on the NBA

data (Figure 6.4b). For NBA, the average number of tuples per player is over 300. In contrast, the average number of tuples per author is below 2. Even the most renowned researchers have tuples for fewer than 40 years. The consequence of a small sample rate is a high probability of missing all tuples of an author, even for the most prolific ones. A second set of results shown in Figure 6.8a is based on a different sampling method: sample for each object at a rate proportional to the size of the object, while the overall sample rate remains the same. By adopting this sampling strategy, we implicitly assume that points of $\mathcal{S}_{\text{sparse}}$ are likely to be yielded by objects with more tuples, and we get significant improvement over uniform sampling.

The other important factor that influences the effectiveness of $\mathcal{O}^+$ selection is the query type. On the NBA data, we see a notable difference between the recall $\tilde{\mathcal{S}}_{\text{sparse}}$ for count query and the other two query types. Similarly, on the WIKI data, we observe a notable difference between *streak* query and the other two types (Figure 6.9). For *streak* query, a point of $\mathcal{S}_{\text{sparse}}$ comes from a (possibly small) number of consecutive tuples. Uniformly sampled tuples may not be representative at all for the overall quality of an object w.r.t. a streak query. Similarly, for *projection* query, say for NBA, a generally lousy player could have one outstanding game that contributes to $\mathcal{S}_{\text{sparse}}$, which will unlikely be picked up by a small uniform sample. On the contrary, any point in $\mathcal{S}$ of a *count* query considers all tuples of an object, making $\mathcal{O}^+$ selection less vulnerable to uniform samples. Had Algorithm 12 known the behavior of the query function f, smarter sampling strategies could be deployed to better guide the selection of $\mathcal{O}^+$.

6.6.3 Efficiency

We evaluate the efficiency of the sampling-based algorithm when varying various parameters, namely (i) the object selection strategy, (ii) the distance metric (L_∞-/L_1-/L_2-norm), and (iii) the computation budget η. The efficiency is measured as

execution-time speed-up over the baseline algorithm (Algorithm 11), which performs full evaluation on all objects. Not surprisingly, the object selection strategy and the distance metric do not influence the efficiency of Algorithm 12. Hence, the results are omitted.

Varying ζ, η^+, and η^-. Figure 6.10 shows the speed-up of the sampling-based algorithm over the baseline, varying ζ, η^+, and η^-, respectively, with the other two fixed. The solid *optimal* curve shows the maximum possible speed-up, i.e. $(\zeta + \eta^+ + \eta^-)^{-1}$, for linear-time query function f, excluding any overhead during object selection. Roughly the same amount of speed-up is observed for all three query types, regardless of the size and distribution of the input data. Results on DBLP and WIKI data are similar and thus omitted.

6.7 Related Works and Discussion

Visualization-assisted data analysis has recently attracted a lot of attention in both database and CHI communities. Jugel et al. (2014) studied data reduction techniques for high-volume time series data driven by visualization constraints, while our work is motivated by limit on data access. Along the same line, the idea of data reduction for efficient visualization of time series data had previously been explored by (Burtini et al., 2013).

The problem of rapid generation of approximate visualizations while preserving crucial properties was studied by Kim et al. (2014) for bar chart, which employs very different techniques. Efficiency-precision tradeoff for exploratory visualization of large dataset has also been studied in the CHI community (Fisher, 2011; Fisher et al., 2012). The idea of using prefetching techniques to improve visualization efficiency for real-time responses in an interactive environment has been studied by Doshi et al. (2003). Kandel et al. (2012) presented *Profiler* for visualization-assisted

data exploration and anomaly detection from an HCI perspective.

In this chapter, we have considered outliers as points having a small number of neighbors (thus not necessarily on the skyline), but the idea of finding global outliers from promising objects by evaluating sample data can apply to outliers of other forms, such as skyline points (Börzsönyi et al., 2001). The sketching algorithm using points of $\mathcal{S}^+ \backslash \tilde{\mathcal{S}}_{\text{sparse}}$ along with $\mathcal{S}^-$ with estimated counts can work as is.

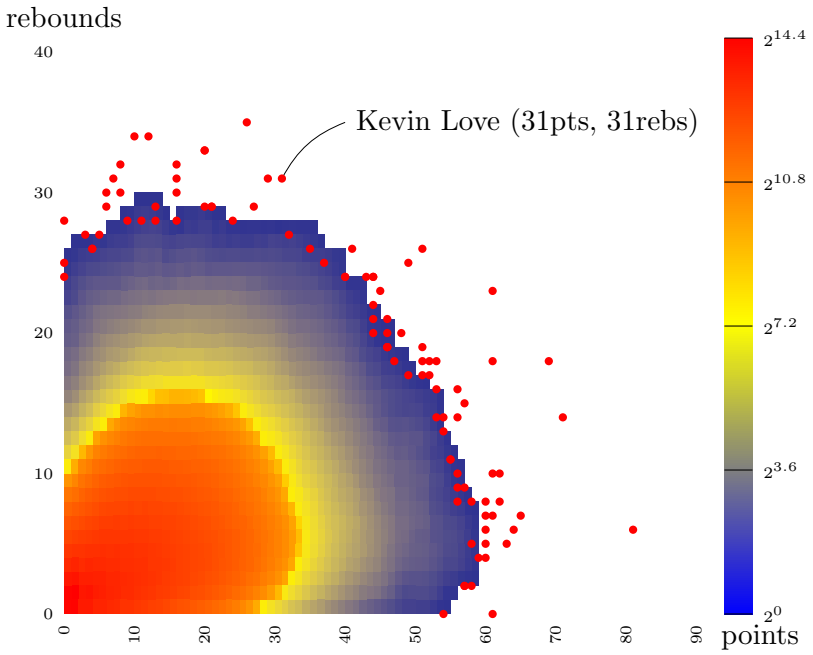
The type of uniqueness-based fact-finding has been studied for skyline and sky-band points in Chapter 5, where specialized algorithms were proposed to efficiently find all interesting and unique points of a large point set; in this chapter, we propose a neighborhood sparsity based uniqueness definition and propose an algorithm tailoring towards the visualization method instead of the claim type.

The idea behind the two-phase sampling-based algorithm is related to the notion of *coreset* (Agarwal et al., 2005) in computational geometry. Since we do not have direct access to the result set $\mathcal{S}$, for each query instance f, we construct "coresets" of objects $\mathcal{O}^+$ and $\mathcal{O}^-$ instead of coresets of points. The selection of $\mathcal{O}^-$ using random sampling is analogous to drawing random samples from the point set (Vapnik and Chervonenkis, 1971; Phillips, 2012). On the other hand, the selection of $\mathcal{O}^+$ is optimized towards preserving another extent measure: minimum density points of $\mathcal{S}$.

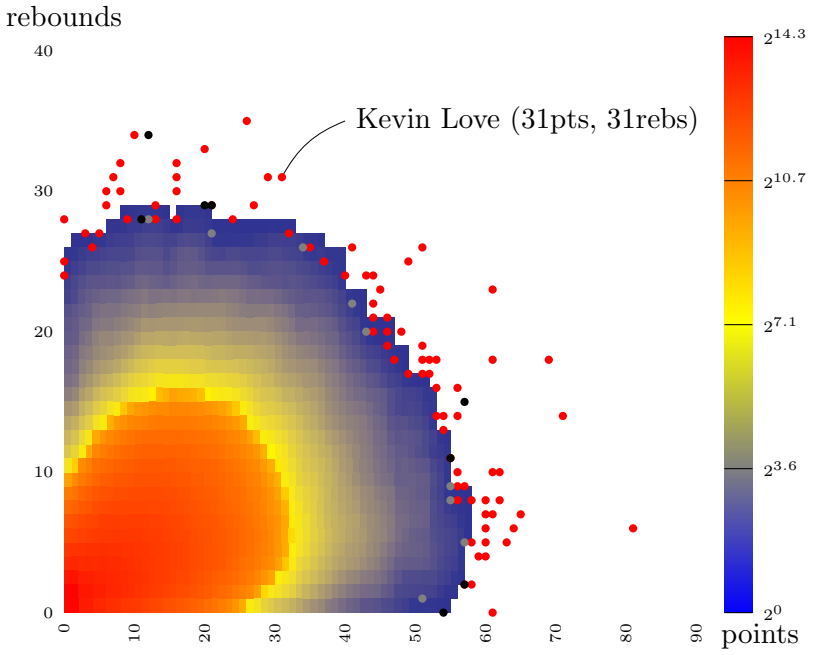
Experiments in this chapter are all conducted in memory. However, at service time, to support exploration query evaluation for many datasets, and to avoid dedicating the memory to a single dataset, objects' full data is hosted in SSTable (Chang et al., 2008) and brought into memory only upon request via the SSTable service API. Data accessing cost will become even more dominant in the execution time. In that case, our sampling-based algorithm with a data access budget would increase its advantage over the baseline algorithm. Also, we have not explored the possibility of parallel evaluation of query function f. On large data sets, parallel query evaluation for different objects will further speed up the overall efficiency.

6.8 Conclusion

In journalism, claims derived from data are important ingredients for many stories, ranging from politics to sports. A common analysis for determining the quality of a claim is to compare it with other claims of the same form. Such exploratory analysis can usually be carried out effectively via visualization. In this chapter, we consider claims that can be modeled as queries whose results can be represented as 2D points, and we focus on one common type of visualization—a combination of 2d scatter plot for outliers and a heatmap for overall distribution. We propose an efficient two-phase sampling-based algorithm that works with any query function. The algorithm first executes the query function on a sample of the dataset, and then, based on the result, further selects additional data to access in order to produce a final approximate answer. Experiments show that our algorithm is efficient and is able to preserve result properties important to visualization—namely the outliers and the overall distribution.



(a) $\mathcal{S}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ from full result.



(b) $\tilde{\mathcal{S}}_{\text{sparse}}$ and $\mathcal{S}_{\text{sketch}}$ from an approximate solution: ● true positive; ● false positive; ● false negative (all w.r.t. $\mathcal{S}_{\text{sparse}}$).

FIGURE 6.1: Projection query example on (`points`, `rebounds`), visualized using 2d scatter plot for $\mathcal{S}_{\text{sparse}}$ and $\tilde{\mathcal{S}}_{\text{sparse}}$, on top of heatmap for $\mathcal{S}_{\text{sketch}}$. In the heatmap, the weight of each point is distributed uniformly in its neighborhood.

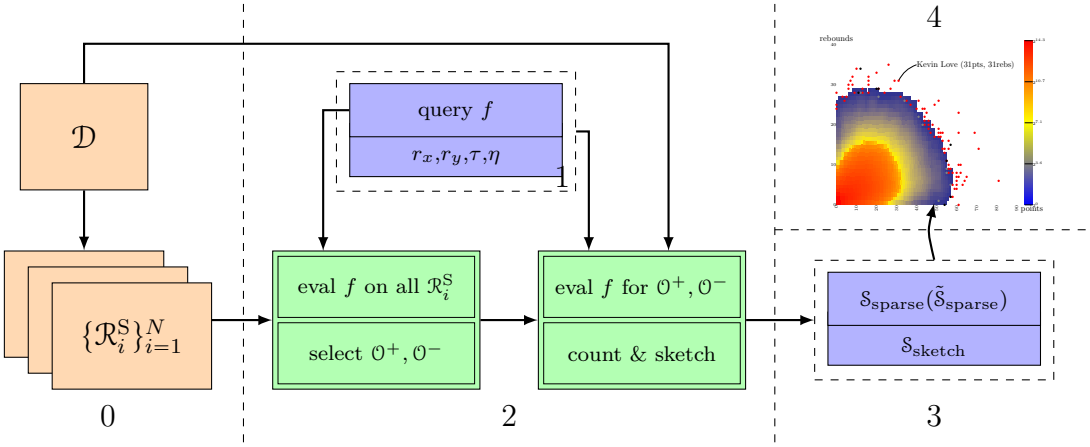
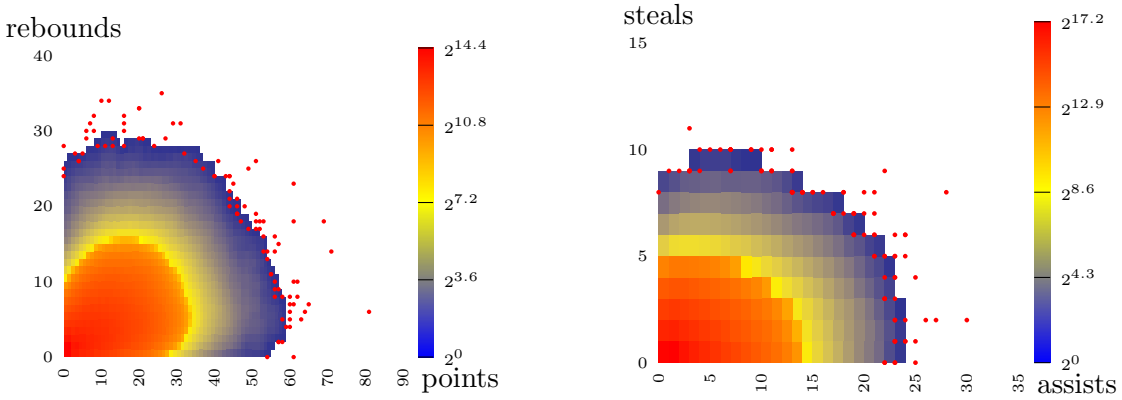
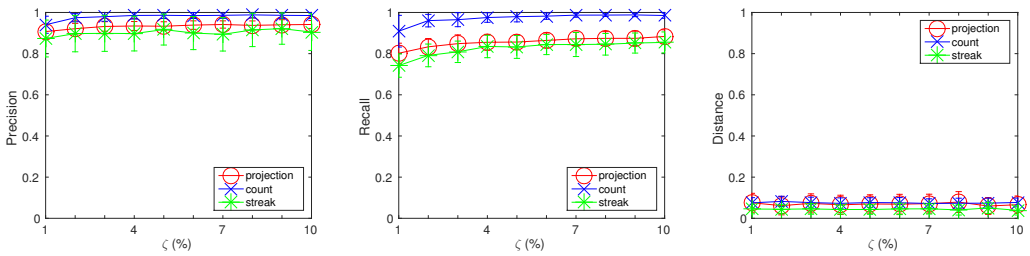


FIGURE 6.2: System workflow.



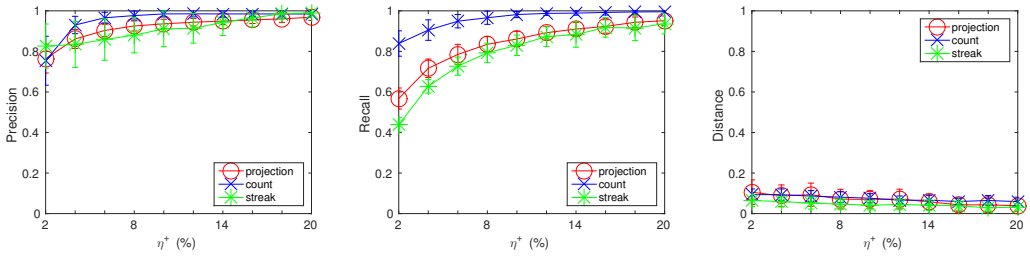
(a) f_1 : Projection on `points-rebounds` plain (b) f_2 : Projection on `assists-steals` plain

FIGURE 6.3: Comparing results of projection queries with different attributes.



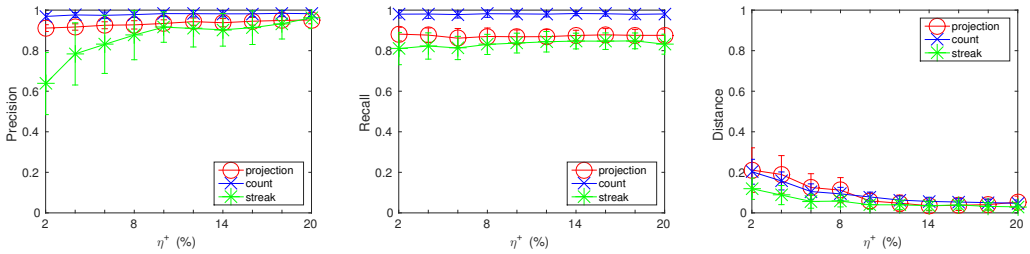
(a) Precision (b) Recall (c) Sketch Distance

FIGURE 6.4: **(NBA)** Vary sample rate ζ between 1-10%. Fix $\eta^+ = \eta^- = .1$.



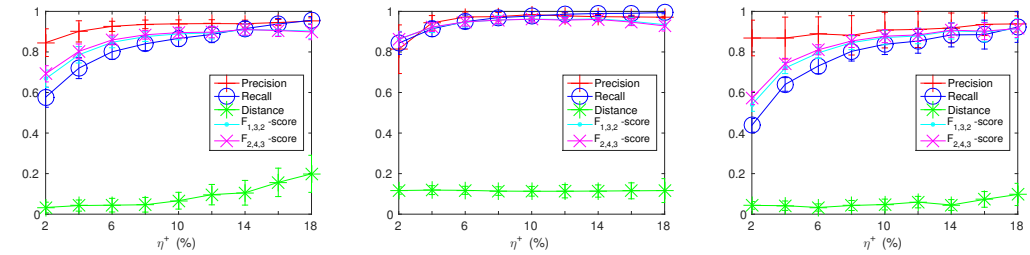
(a) Precision (b) Recall (c) Sketch Distance

FIGURE 6.5: **(NBA)** Vary budget η^+ for $\mathcal{O}^+$ between 2-20%. Fix $\zeta = .05$, $\eta^- = .1$.



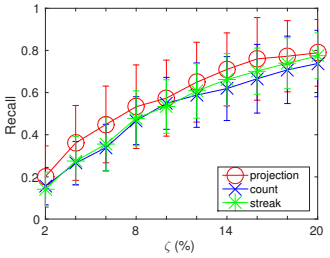
(a) Precision (b) Recall (c) Sketch Distance

FIGURE 6.6: **(NBA)** Vary budget η^- for $\mathcal{O}^-$ between 2-20%. Fix $\zeta = .05$, $\eta^+ = .1$.

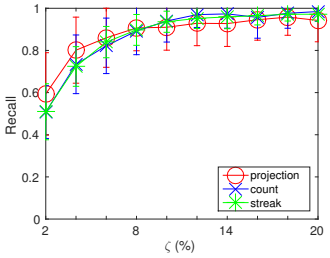


(a) Projection (b) Count (c) Streak

FIGURE 6.7: **(NBA)** Fix total budget $\eta = .2$ for $\mathcal{O}^+$ and $\mathcal{O}^-$, vary η^+ for $\mathcal{O}^+$ between 2-18%.



(a) Uniform sample



(b) Sample rate proportional to objects' data size

FIGURE 6.8: **(DBLP)** Compare recall of $\tilde{\mathcal{S}}_{\text{sparse}}$ with two different sampling strategies, at varying sample rate 1-10%, fixing $\eta^+ = \eta^- = .1$.

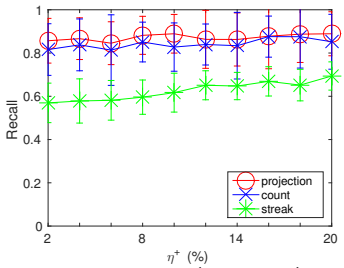
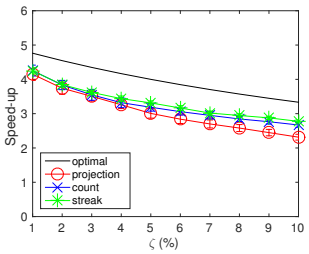
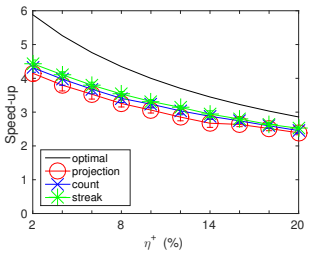


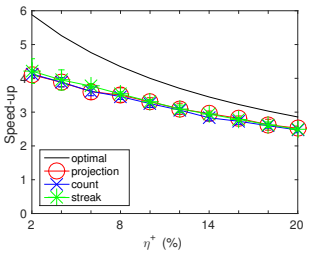
FIGURE 6.9: **(WIKI)** Recall of $\tilde{\mathcal{S}}_{\text{sparse}}$. Vary η^+ 2-20%. Fix $\zeta = .05, \eta^- = .1$



(a) Vary ζ. Fix $\eta^+ = \eta^- = .1$.



(b) Vary η^+. Fix $\zeta = .05$, $\eta^- = .1$.



(c) Vary η^-. Fix $\zeta = .05$, $\eta^+ = .1$.

FIGURE 6.10: **(NBA)** Efficiency of algorithms varying sample rate and budget.

7

Conclusion

Many interesting and practical questions arise from data journalism. Specifically, this dissertation tackles a set of problems concerning claims based on structured data: finding counter-arguments for questionable claims, reverse-engineering vague claims, finding good leads from a data set, looking for claims with specific quality, visualizing claims for outlier detection. An underlying idea that unifies the solutions to all these problems is to regard claims as queries with parameters. This idea led us to a powerful framework for modeling these problems as computational ones, and opened up opportunities for solving them by developing efficient algorithms. We have deveoped a system prototype (iCheck) implementing the fact-checking and lead-finding tasks studied in this dissertation.

Our proposed framework and the work around it has opened up more research problems than they answer, some of which already discussed in Section 3.5. To have hope for a fully automatic solution to fact-checking and lead-finding, obtaining high-quality strucutred data is the first step. How to use claim context to help determine relevant data sources (Rahm and Bernstein, 2001)? How to assess the quality of available data sources (Balakrishnan and Kambhampati, 2011)? How to deal with

conflict information from different data sources (Dong et al., 2009; Zhao et al., 2012)?

If the data is uncertain, how does this uncertainty affect the quality of claims? Can we design a unified model for consider joint perturbation of parameter and data, and design efficient algorithms for the fact-checking and lead-finding tasks? How do we characterize the influence of data uncertainty on claim quality? How do we prioritize expensive data cleansing operations when a budget is given?

Another pre-requisite to our problems, is the translation of a claim in natural language to a mathematical function as our claim template. This step in the analysis pipeline can benefit from advances in research in *natural language querying* (Popescu et al., 2003; Li and Jagadish, 2014) and crowdsourcing (Franklin et al., 2011). Instead of asking the crowd to answer queries, what is the most efficient and effective way of asking the crowd to write queries for us?

Besides the claim quality measures we have already studied in this dissertation, what are the other measures and how do we support them efficiently? For example, if the underlying data is uncertain, can we measure the quality of a claim by its certainty? If the underlying data is evolving, can we measure the quality of a claim by its "durability," i.e., how long it is expected to be true?

Evaluation remains a challenging problem for many aspects of our work, as it is hard to define the "ground truth" objectively: What is a good counter-argument? What is an interesting lead? While we have evaluated the effectiveness of our techniques by carefully analyzing their results on a case-by-case basis, a more thorough evaluation would require user studies at a much larger scale. Furthermore, recognizing that different users may have different peceptions of result quality, we may be able to track users' interactions with the system and optimize for click-through rate on items recommended by our system. The same idea can be used to simplify the work in setting up fact checking and lead finding for a new domain. By observing user responses to a series of questions posed by our system, we can learn functions

required by our computational framework, such as the result strength and parameter sensibility functions.

To conclude, this dissertation has taken the initial steps towards an end-to-end system empowering journalists and the public to combat the "lies, d—ed lies, and statistics" that permeate our public life today. While we are far from the "holy grail" of fully automatic fact-checkers and lead-finders, this dissertation has shown considerable promise of a computational approach in assisting human journalists in tasks for which automation has been previously considered impossible.

Appendix A

Appendix for Chapter 6

Following the notations of Section 6.5.2, we derive the bounds on $\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}})$ here.

First, by linearity of expectation, we have

$$\mu_j = \mathbb{E}[Z_j] = |\mathcal{S} \cap P_j|,\ \mu_Z = \mathbb{E}[Z] = |\mathcal{S}|.$$

It is easy to see the following bound on $\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}})$:

$$\delta(\mathcal{S}, \mathcal{S}_{\text{sketch}}) \leqslant 1 - \frac{\sum_j \min\{Z_j, |\mathcal{S} \cap P_j|\}}{\max\{Z, |\mathcal{S}|\}} = 1 - \frac{\sum_j \min\{Z_j, \mu_j\}}{\max\{Z, \mu_Z\}}.$$

By the (one-sided) Chebyshev's inequality, we have

$$\begin{aligned}
&\mathbf{Pr}\left[\sum_j \min\{Z_j, \mu_j\} \geqslant (1-\Delta^-)\mu_Z\right] \\
\geqslant &\mathbf{Pr}\left[\bigcap_j Z_j \geqslant (1-\Delta^-)\mu_j\right] \\
\geqslant &1 - \mathbf{Pr}\left[\bigcup_j Z_j \leqslant (1-\Delta^-)\mu_j\right] \\
\geqslant &1 - \sum_j \frac{1}{1 + (\Delta^-\mu_j/\sigma_j)^2}
\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}\left[\max\{Z,\mu_Z\} \geqslant (1+\Delta^+)\mu_Z\right] &= \mathbf{Pr}\left[Z \geqslant (1+\Delta^+)\mu_Z\right] \\ &\leqslant \frac{1}{1+(\Delta^+\mu_Z/\sigma_Z)^2}.\end{aligned}$$

Combining the two inequalities above, we have

$$\begin{aligned}&\mathbf{Pr}\left[\delta(\mathcal{S},\mathcal{S}_{\text{sketch}}) \geqslant 1-\frac{1-\Delta^-}{1+\Delta^+}\right] \\ \leqslant&\mathbf{Pr}\left[\sum_j \min\{Z_j,\mu_j\} \leqslant (1-\Delta^-)\mu_Z \vee Z \geqslant (1+\Delta^-)\mu_Z\right] \\ \leqslant&\sum_j \frac{1}{1+(\Delta^-\mu_j/\sigma_j)^2} + \frac{1}{1+(\Delta^+\mu_Z/\sigma_Z)^2}.\end{aligned}$$

To see the bounds dependent on correlation of point distribution by objects, let c_{ij} denote the number of points in partition P_j that come from object i; i.e., $c_{ij} = |f(\mathcal{R}_i) \cap P_j|$, and $C_i = \sum_j c_{ij} = |f(\mathcal{R}_i)|$. It follows that $Z_j = \lambda \cdot \sum_i X_i c_{ij}$. We have, for each i:

$$\mu_i = \sum_j c_{ij}, \quad \sigma_i^2 = \frac{1-p}{p} \cdot \sum_i c_{ij}^2,$$

and for Z,

$$\mu_Z = \sum C_j, \quad \sigma_Z^2 = \frac{1-p}{p} \cdot \sum_i C_i^2.$$

For any two partitions j and j', the covariance and correlation between Z_j and $Z_{j'}$ are given by

$$\sigma_{jj'} = \frac{1-p}{p} \cdot \sum_i c_{ij}c_{ij'}, \quad \rho_{jj'} = \frac{\sigma_{jj'}}{\sigma_j\sigma_{j'}}.$$

The tighter bound can be obtained by taking into account the correlation among

Z_j's, using the dependent multi-variate Chebyshev's inequality as follows:

$$\begin{aligned}
&\mathbf{Pr}\left[\bigcap_j Z_j \geqslant (1-\Delta^-)\mu_j\right] \\
\leqslant &1 - \frac{1}{m^2}\left(\sqrt{u} + \sqrt{m-1}\sqrt{\frac{m}{(\Delta^-)^2}\sum_j \frac{\sigma_j^2}{\mu_j^2} - u}\right)^2,
\end{aligned}$$

where

$$u = \frac{1}{(\Delta^-)^2}\sum_j \sum_{j'} \frac{\rho_{jj'}}{\mu_j \mu_{j'}}.$$

Setting $\rho_{jj'} = 1$ for all j, j', and we get Ineq. (6.7), (6.8), and (6.9).

Bibliography

Ackerman, M., Ben-David, S., Brânzei, S., and Loker, D. (2012), "Weighted Clustering," in *Proceedings of the 2012 National Conference on Artificial Intelligence*, pp. 858–863, Toronto, Ontario, Canada.

Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R. (2005), "Geometric Approximation via Coresets," *Combinatorial and Computational Geometry*, 52, 1–30.

Aggarwal, C. C. (ed.) (2009), *Managing and Mining Uncertain Data*, Springer.

Andrew, A. M. (1979), "Another Efficient Algorithm for Convex Hulls in Two Dimensions," *Information Processing Letters*, 9, 216–219.

Aurenhammer, F. and Klein, R. (2000), "Voronoi Diagrams," *Handbook of Computational Geometry*, 5, 201–290.

Balakrishnan, R. and Kambhampati, S. (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*, pp. 227–236, Hyderabad, India.

Bentley, J., Kung, H., Schkolnick, M., and Thompson, C. (1978), "On the Average Number of Maxima in a Set of Vectors and Applications," *Journal of the ACM*, 25, 536–543.

Berg, M. D., Kreveld, M. V., Overmars, M., and Schwarzkopf, O. C. (2000), *Computational Geometry*, Springer.

Bernstein, P. A. and Haas, L. M. (2008), "Information Integration in the Enterprise," *Communications of the ACM*, 51, 72–79.

Beyer, K. and Ramakrishnan, R. (1999), "Bottom-up computation of sparse and Iceberg CUBE," *ACM SIGMOD Record*, 28, 359–370.

Börzsönyi, S., Kossmann, D., and Stocker, K. (2001), "The Skyline Operator," in *Proceedings of the 2001 International Conference on Data Engineering*, pp. 421–430, Heidelberg, Germany.

Buchbinder, N., Feldman, M., Naor, J., and Roy, S. (2012), "A Tight Linear Time (1/2)-approximation for Unconstrained Submodular Maximization," in *Proceedings of the 2012 IEEE Symposium on Foundations of Computer Science*, pp. 649–658, IEEE.

Buchta, C. (1989), "On the Average Number of Maxima in a Set of Vectors," *Information Processing Letters*, 33, 63–65.

Burtini, G., Fazackerley, S., and Lawrence, R. (2013), "Time Series Compression for Adaptive Chart Generation," in *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*, pp. 1–6, IEEE.

Carbonell, J. and Goldstein, J. (1998), "The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries," in *Proceedings of the 1998 ACM SIGIR International Conference on Research and Development in Information Retrieval*, pp. 335–336, Berkeley, California, USA.

Catallo, I., Ciceri, E., Fraternali, P., Martinenghi, D., and Tagliasacchi, M. (2013), "Top-k Diversity Queries over Bounded Regions," *ACM Transactions on Database Systems*, 38.

Chan, T. F. and Vese, L. (2001), "Active Contours without Edges," *IEEE Transactions on Image Processing*, 10, 266–277.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008), "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems*, 26, 4.

Chazelle, B. (1988), "A Functional Approach to Data Structures and its Use in Multidimensional Searching," *SIAM Journal on Computing*, 17, 427–462.

Chen, D. Z. and Wang, H. (2011), "Efficient Algorithms for the Weighted k-Center Problem on a Real Line," in *Proceedings of the 2011 International Symposium on Algorithms and Computation*, pp. 584–593, Yokohama, Japan.

Chomicki, J., Godfrey, P., Gryz, J., and Liang, D. (2003), "Skyline with Presorting," in *Proceedings of the 2003 International Conference on Data Engineering*, pp. 711–719, Bangalore, India.

Cohen, S., Hamilton, J. T., and Turner, F. (2011a), "Computational Journalism," *Communications of the ACM*, 54, 66–71.

Cohen, S., Li, C., Yang, J., and Yu, C. (2011b), "Computational Journalism: A Call to Arms to Database Researchers," in *Proceedings of the 2011 Conference on Innovative Data Systems Research*, pp. 148–151, Asilomar, California, USA.

D., H., Darera, P. N., and Haritsa, J. R. (2008), "Identifying Robust Plans through Plan Diagram Reduction," in *Proceedings of the 2008 International Conference on Very Large Data Bases*, pp. 1124–1140, Auckland, New Zealand.

Dalvi, N. N., Ré, C., and Suciu, D. (2009), "Probabilistic Databases: Diamonds in the Dirt," *Communications of the ACM*, 52, 86–94.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977), "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society Series B (methodological)*, 39, 1–38.

Doan, A., Halevy, A., and Ives, Z. (2012), *Principles of Data Integration*, Morgan Kaufmann, 1 edn.

Dong, X. L., Berti-Equille, L., and Srivastava, D. (2009), "Integrating Conflicting Data: The Role of Source Dependence," *Proceedings of the VLDB Endowment*, 2, 550–561.

Doshi, P. R., Rundensteiner, E. A., and Ward, M. O. (2003), "Prefetching for Visual Data Exploration," in *Proceedings of the 2003 International Conference on Database Systems for Advanced Applications*, pp. 195–202, Kyoto, Japan, IEEE.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. (2001), "Rank Aggregation Methods for the Web," in *Proceedings of the 2001 International Conference on World Wide Web*, pp. 613–622, Hong Kong, China.

Fagin, R., Lotem, A., and Naor, M. (2003), "Optimal Aggregation Algorithms for Middleware," *Journal of Computer and System Sciences*, 66, 614–656.

Fisher, D. (2011), "Incremental, Approximate Database Queries and Uncertainty for Exploratory Visualization," in *Proceedings of the 2011 Large Data Analysis and Visualization*, pp. 73–80, Paris, France, IEEE.

Fisher, D., Popov, I., Drucker, S., and mc schraefel (2012), "Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster," in *Proceedings of the 2012 International Conference on Human Factors in Computing Systems*, pp. 1673–1682, Austin, Texas, USA, ACM.

Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., and Xin, R. (2011), "CrowdDB: Answering Queries with Crowdsourcing," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pp. 61–72, Athens, Greece, ACM.

Ganguly, S. (1998), "Design and Analysis of Parametric Query Optimization Algorithms," in *Proceedings of the 1998 International Conference on Very Large Data Bases*, pp. 228–238, New York City, New York, USA.

Giles, J. (2012), "Truth Goggles," *The New Scientist*, pp. 44–47.

Godfrey, P., Shipley, R., and Gryz, J. (2005), "Maximal Vector Computation in Large Data Sets," in *Proceedings of the 2005 International Conference on Very Large Data Bases*, pp. 229–240, Trondheim, Norway.

Gray, J., Bosworth, A., Layman, A., and Pirahesh, H. (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*, pp. 152–159, New Orleans, Louisiana, USA.

Guestrin, C., Krause, A., and Singh, A. P. (2005), "Near-optimal Sensor Placements in Gaussian Processes," in *Proceedings of the 2005 International Conference on Machine Learning*, pp. 265–272, Bonn, Germany.

Harel, D. and Tarjan, R. E. (1984), "Fast Algorithms for Finding Nearest Common Ancestors," *SIAM Journal on Computing*, 13, 338–355.

Haritsa, J. R. (2009), "The KNDN Problem: A Quest for Unity in Diversity," *IEEE Data Engineering Bulletin*, 32, 15–22.

Hasan, M., Kashyap, A., Hristidis, V., and Tsotras, V. J. (2014), "User Effort Minimization Through Adaptive Diversification," in *Proceedings of the 2014 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 203–212, New York City, New York, USA.

Hassan, N., Sultana, A., Wu, Y., Zhang, G., Li, C., Yang, J., and Yu, C. (2014), "Data In, Fact Out: Automated Monitoring of Facts by FactWatcher. Demonstration description," *Proceedings of the VLDB Endowment*, 7, 1557–1560.

He, Z. and Lo, E. (2012), "Answering Why-not Questions on Top-k Queries," in *Proceedings of the 2012 International Conference on Data Engineering*, pp. 750–761, Washington DC, USA.

Hulgeri, A. and Sudarshan, S. (2003), "AniPQO: Almost Non-intrusive Parametric Query Optimization for Nonlinear Cost Functions," in *Proceedings of the 2003 International Conference on Very Large Data Bases*, pp. 766–777, Berlin, Germany.

Ioannidis, Y. E., Ng, R. T., Shim, K., and Sellis, T. K. (1992), "Parametric Query Optimization," in *Proceedings of the 1992 International Conference on Very Large Data Bases*, pp. 103–114, Vancouver, Canada.

Jain, A., Sarda, P., and Haritsa, J. R. (2004), "Providing Diversity in K-Nearest Neighbor Query Results," in *Proceedings of the 2004 Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 404–413, Sydney, Australia.

Jampani, R., Xu, F., Wu, M., Perez, L. L., Jermaine, C., and Haas, P. J. (2011), "The Monte Carlo Database System: Stochastic Analysis Close to the Data," *ACM Transactions on Database Systems*, 36, 18.

Jiang, X., Li, C., Luo, P., Wang, M., and Yu, Y. (2011), "Prominent Streak Discovery in Sequence Data," in *Proceedings of the 2011 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1280–1288, San Diego, California, USA.

Jugel, U., Jerzak, Z., Hackenbroich, G., and Markl, V. (2014), "M4: a Visualization-oriented Time Series Data Aggregation," *Proceedings of the VLDB Endowment*, 7, 797–808.

Kandel, S., Parikh, R., Paepcke, A., Hellerstein, J. M., and Heer, J. (2012), "Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment," in *Proceedings of the 2012 International Conference on Advanced Visual Interfaces*, pp. 547–554, Capri Island, Naples, Italy, ACM.

Kim, A., Blais, E., Parameswaran, A., Indyk, P., Madden, S., and Rubinfeld, R. (2014), "Rapid Sampling for Visualizations with Ordering Guarantees," *arXiv preprint arXiv:1412.3040*.

Kimmel, R. (2003), "Fast Edge Integration," in *Geometric Level Set Methods in Imaging, Vision, and Graphics*, pp. 59–77, Springer.

Kimmel, R. and adn Alfred M Bruckstein, A. A. (1995), "Finding Shortest Paths on Surfaces Using Level Sets Propagation," *(IEEE) Transactions on Pattern Analysis and Machine Intelligence*, 17, 635–640.

Kimmel, R. and Bruckstein, A. M. (2003), "Regularized Laplacian Zero Crossings as Optimal Edge Integrators," *International Journal of Computer Vision*, 53, 225–243.

Kimmel, R. and Sethian, J. A. (1998), "Computing Geodesic Paths on Manifolds," *Proceedings of the National Academy of Sciences*, 95, 8431–8435.

Kossmann, D., Ramsak, F., and Rost, S. (2002), "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," in *Proceedings of the 2002 International Conference on Very Large Data Bases*, pp. 275–286, Hong Kong, China.

Krause, A. and Guestrin, C. (2008), "Beyond convexity: Submodularity in Machine Learning," *ICML Tutorials*.

Kung, H.-T., Luccio, F., and Preparata, F. P. (1975), "On Finding the Maxima of a Set of Vectors," *Journal of the ACM*, 22, 469–476.

Li, F. and Jagadish, H. (2014), "Constructing an Interactive Natural Language Interface for Relational Databases," *Proceedings of the VLDB Endowment*, 8, 73–84.

Li, X., Meng, W., and Yu, C. T. (2011), "T-verifier: Verifying Truthfulness of Fact Statements," in *Proceedings of the 2011 International Conference on Data Engineering*, pp. 63–74, Hannover, Germany.

Li, Y., Yang, H., and Jagadish, H. V. (2006), "Constructing a Generic Natural Language Interface for an XML Database," in *Proceedings of the 2006 International Conference on Extending Database Technology*, pp. 737–754, Munich, Germany.

Li, Y., Chaudhuri, I., Yang, H., Singh, S., and Jagadish, H. V. (2007), "DaNaLIX: A Domain-Adaptive Natural Language Interface for Querying XML," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pp. 1165–1168, Beijing, China.

Lin, H. and Bilmes, J. (2011), "A Class of Submodular Functions for Document Summarization," in *Proceedings of the 2011 Annual Meeting of the Association for Computational Linguistics*, pp. 510–520, Association for Computational Linguistics.

Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. (2007), "Selecting Stars: The k Most Representative Skyline Operator," in *Proceedings of the 2007 International Conference on Data Engineering*, pp. 86–95, Istanbul, Turkey.

Lin, X., Mukherji, A., Rundensteiner, E. A., Ruiz, C., and Ward, M. O. (2013), "PARAS: A Parameter Space Framework for Online Association Mining," *Proceedings of the VLDB Endowment*, 6, 193–204.

Lloyd, S. P. (1982), "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, 28, 129–137.

Lu, H., Jensen, C. S., and Zhang, Z. (2010), "Flexible and Efficient Resolution of Skyline Query Size Constraints," *IEEE Transactions on Knowledge and Data Engineering*, 23, 991–1005.

MacQueen, J. (1967), "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the 1967 Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, Oakland, CA, USA.

Mehlhorn, K. and Näher, S. (1990), "Dynamic Fractional Cascading," *Algorithmica*, 5, 215–241.

Mobahi, H., Rao, S. R., Yang, A. Y., Sastry, S. S., and Ma, Y. (2011), "Segmentation of Natural Images by Texture and Boundary Compression," *International Journal of Computer Vision*, 95, 86–98.

Mouratidis, K. and Pang, H. (2012), "Computing Immutable Regions for Subspace Top-k Queries," *Proceedings of the VLDB Endowment*, 6, 73–84.

Mumford, D. and Shah, J. (1989), "Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems," *Communications on Pure and Applied Mathematics*, 42, 577–685.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978), "An Analysis of Approximations for Maximizing Submodular Set FunctionsI," *Mathematical Programming*, 14, 265–294.

Ohlander, R., Price, K., and Reddy, D. R. (1978), "Picture Segmentation Using a Recursive Region Splitting Method," *Comptuer Graphics and Image Processing*, 8, 313–333.

Papadias, D., Tao, Y., Fu, G., and Seeger, B. (2005), "Progressive Skyline Computation in Database Systems," *ACM Transactions on Database Systems*, 30, 41–82.

Pappas, T. N. (1992), "An Adaptive Clustering Algorithm for Image Segmentation," *IEEE Transactions on Signal Processing*, 40, 901–914.

Pei, J., Yuan, Y., Lin, X., Jin, W., Ester, M., Liu, Q., Wang, W., Tao, Y., Yu, J. X., and Zhang, Q. (2006), "Towards Multidimensional Subspace Skyline Analysis," *ACM Transactions on Database Systems*, 31, 1335–1381.

Phillips, J. M. (2012), "Chernoff-Hoeffding Inequality and Applications," *arXiv preprint arXiv:1209.6396*.

Popescu, A.-M., Etzioni, O., and Kautz, H. A. (2003), "Towards a Theory of Natural Language Interfaces to Databases," in *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, pp. 149–157, Miami, Florida, USA.

Qin, L., Yu, J. X., and Chang, L. (2012), "Diversifying Top-K Results," in *Proceedings of the 2012 International Conference on Very Large Data Bases*, pp. 1124–1135, Istanbul, Turkey.

Quinn, A. J. and Bederson, B. B. (2011), "Human Computation: A Survey and Taxonomy of a Growing Field," in *Proceedings of the 2011 International Conference on Human Factors in Computing Systems*, pp. 1403–1412, Vancouver, British Columbia, Canada.

Rahm, E. and Bernstein, P. A. (2001), "A Survey of Approaches to Automatic Schema Matching," *The VLDB Journal*, 10, 334–350.

Rao, S. R., Mobahi, H., Yang, A. Y., Sastry, S. S., and Ma, Y. (2010), "Natural Image Segmentation with Adaptive Texture and Boundary Encoding," in *Proceedings of the 2009 Asian Conference on Computer Vision*, vol. 1, pp. 135–146, Xi'an, China, Springer.

Roy, S. and Suciu, D. (2014), "A Formal Approach to Finding Explanations for Database Queries," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 1579–1590, Snowbird, Utah, USA.

Rubner, Y., Tomasi, C., and Guibas, L. J. (1998), "A Metric for Distributions with Applications to Image Databases," in *Proceedings of the 1998 IEEE International Conference on Computer Vision*, pp. 59–66, Bombay, India, IEEE.

Sarma, A. D., Parameswaran, A. G., Garcia-Molina, H., and Widom, J. (2010), "Synthesizing View Definitions from Data," in *Proceedings of the 2010 International Conference on Database Theory*, pp. 89–103, Lausanne, Switzerland.

Schrijver, A. (2003), *Combinatorial Optimization: Polyhedra and Efficiency*, vol. 24, Springer.

Soliman, M. A., Ilyas, I. F., Martinenghi, D., and Tagliasacchi, M. (2011), "Ranking with Uncertain Scoring Functions: Semantics and Sensitivity Measures," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pp. 805–816, Athens, Greece.

Tan, K.-L., Eng, P.-K., and Ooi, B. C. (2001), "Efficient Progressive Skyline Computation," in *Proceedings of the 2001 International Conference on Very Large Data Bases*, pp. 301–310, Roma, Italy.

Tao, Y., Xiao, X., and Pei, J. (2006), "Subsky: Efficient Computation of Skylines in Subspaces," in *Proceedings of the 2006 International Conference on Data Engineering*, pp. 65–65, Atlanta, Georgia, USA.

Tarjan, R. E. (1979), "Applications of Path Compression on Balanced Trees," *Journal of the ACM*, 26, 690–715.

Tran, Q. T. and Chan, C.-Y. (2010), "How to ConQueR Why-Not Questions," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pp. 15–26, Indianapolis, Indiana, USA.

Tran, Q. T., Chan, C.-Y., and Parthasarathy, S. (2009), "Query by Output," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pp. 535–548, Providence, Rhode Island, USA.

Tsaparas, P., Palpanas, T., Kotidis, Y., Koudas, N., and Srivastava, D. (2003), "Ranked Join Indices," in *Proceedings of the 2003 International Conference on Data Engineering*, pp. 277–288, Bangalore, India.

Tschiatschek, S., Iyer, R. K., Wei, H., and Bilmes, J. A. (2014), "Learning Mixtures of Submodular Functions for Image Collection Summarization," in *Proceedings of the 2014 Advances in Neural Information Processing Systems*, pp. 1413–1421, Montreal, Canada.

Vapnik, V. N. and Chervonenkis, A. Y. (1971), "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory of Probability and Its Applications*, 16, 264–280.

Vlachou, A., Doulkeridis, C., Norvag, K., and Vazirgiannis, M. (2008), "Skyline-based peer-to-peer top-k query processing," in *Proceedings of the 2008 International Conference on Data Engineering*, pp. 1421–1423, Cancun, Mexico.

Voronoï, G. (1908), "Nouvelles applications des paramètres continus à la théorie des formes quadratiques," *Journal für die reine und angewandte Mathematik*, 1908, 97–102.

Walenz, B., Wu, Y., Song, S., Sonmez, E., Wu, E., Wu, K., Agarwal, P. K., Yang, J., Hassan, N., Sultana, A., Zhang, G., Li, C., and Yu, C. (2014), "Finding, Monitoring, and Checking Claims Computationally Based on Structured Data," *Computation+Journalism Symposium*.

Wu, E. and Madden, S. (2013), "Scorpion: Explaining Away Outliers in Aggregate Queries," *Proceedings of the VLDB Endowment*, 6, 553–564.

Wu, Y., Agarwal, P. K., Li, C., Yang, J., and Yu, C. (2012), "On "One of the Few" Objects," in *Proceedings of the 2012 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1487–1495, Beijing, China.

Wu, Y., Walenz, B., Li, P., Shim, A., Sonmez, E., Agarwal, P. K., Li, C., Yang, J., and Yu, C. (2014a), "iCheck: Computationally Combatting "Lies, D–ned Lies, and Statistics". Demonstration description," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 1063–1666, Snowbird, Utah, USA.

Wu, Y., Agarwal, P. K., Li, C., Yang, J., and Yu, C. (2014b), "Toward Computational Fact-Checking," *Proceedings of the VLDB Endowment*, 7, 589–600.

Wu, Y., Harb, B., Yang, J., and Yu, C. (2015), "Efficient Evaluation of Object-centric Exploration Queries for Visualization," *To appear in Proceedings of the VLDB Endowment*, 8.

Yamamoto, Y. and Tanaka, K. (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*, pp. 291–305, Poznan, Poland.

Yamamoto, Y., Tezuka, T., Jatowt, A., and Tanaka, K. (2008), "Supporting Judgment of Fact Trustworthiness Considering Temporal and Sentimental Aspects," in *Proceedings of the 2008 International Conference on Web Information Systems Engineering*, pp. 206–220, Auckland, New Zealand.

Yiu, M. L. and Mamoulis, N. (2007), "Efficient Processing of Top-k Dominating Queries on Multi-dimensional Data," in *Proceedings of the 2007 International Conference on Very Large Data Bases*, pp. 483–494, Vienna, Austria.

Young, H. (1975), "Social Choice Scoring Functions," *SIAM Journal on Applied Mathematics*, pp. 824–838.

Yu, A., Agarwal, P. K., and Yang, J. (2012), "Processing a Large Number of Continuous Preference Top-k Queries," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 397–408, Scottsdale, Arizona, USA.

Zhao, B., Rubinstein, B. I. P., Gemmell, J., and Han, J. (2012), "A Bayesian Approach to Discovering Truth from Conflicting Sources for Data Integration," *Proceedings of the VLDB Endowment*, 5, 550–561.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. (2010), "Solving the Apparent Diversity-accuracy Dilemma of Recommender Systems," *Proceedings of the National Academy of Sciences*, 107, 4511–4515.

Biography

You (Will) Wu was born September 23, 1988 in Shanghai, China. He earned a Bachelors of Engineering in Computer Science and Engineering in Computer Science from the Hong Kong University of Science and Technology in May 2010, and a Masters of Science from Duke University in May 2014.

Will was also involved in the development of system prototypes (Wu et al., 2014a; Hassan et al., 2014; Walenz et al., 2014) that were demonstrated at computer science and journalism conferences. The FactWatcher system (Hassan et al., 2014) won the Excellent Demonstration Award at the VLDB 2014 conference.

Will will join the Structured Data research group of Google Inc. in the New York City office after graduation.