On Detecting Cherry-picked Trendlines *

Abolfazl Asudeh
University of Illinois at Chicago
asudeh@uic.edu

H. V. Jagadish
University of Michigan
jag@umich.edu

You (Will) Wu
Google Research
wuyou@google.com

Cong Yu
Google Research
congyu@google.com

ABSTRACT

Poorly supported stories can be told based on data by cherry-picking the data points included. While such stories may be technically accurate, they are misleading. In this paper, we build a system for detecting cherry-picking, with a focus on trendlines extracted from temporal data. We define a *support* metric for detecting such trendlines. Given a dataset and a statement made based on a trendline, we compute a support score that indicates how cherry-picked it is. Studying different types of trendlines and formalizing terms, we propose efficient and effective algorithms for computing the support measure. We also study the problem of discovering the most supported statements. Besides theoretical analysis, we conduct extensive experiments on real-world data, that demonstrate the validity of our proposed techniques.

PVLDB Reference Format:
Abolfazl Asudeh, H. V. Jagadish, You (Will) Wu, Cong Yu. On Detecting Cherry-picked Trendlines. *PVLDB*, 13(6): 939-952, 2020.
DOI: https://doi.org/10.14778/3380750.3380762

1. INTRODUCTION

"A lie which is half a truth is ever the blackest of lies."
– ALFRED, LORD TENNYSON

Fake news is receiving much attention today. Sometimes fake news may be a complete fabrication. More often, it is based on a grain of truth, such as a fact reported out of context or analysis on cherry-picked data [1]. In fact, cherry-picking is prevalent in almost every controversial domain from tax policy to climate change.

Partisans on one side of an argument look for statements they can make about trends that support their position. They would like not to be caught blatantly lying, so they cherry-pick the factual basis for their conclusion. That is, the points based on which a *statement* is made are carefully selected to show a misleading *"trendline"* that is not a "reasonable" representation of the situation. Even though the trendline is not fake, in that it is supported by the selected data points, it is misleading. In this paper, we study such cherry-picked trendlines. But first, let us look at a couple of examples.

*The work of H. V. Jagadish was supported in part by NSF Grants No. 1741022 and 1934565.

Proceedings of the VLDB Endowment, Vol. 13, No. 6
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3380750.3380762

EXAMPLE 1 (NORTHERN HEMISPHERE'S TEMPERATURE). *In [2], John Mason explains how cherry-picking short time-frames can distort the reality of global warming. He uses the monthly climate data of the year 2012 to support the fantasy-like claim that* THE NORTHERN HEMISPHERE SUMMERS ARE COLDER THAN WINTERS. *Such a statement can be made by selecting two specific time/days in summer and winter, as well as specific locations in Northern Hemisphere, and comparing the temperatures of the selected locations in selected time/days: summer was colder than winter in 2012 as, for example, the (average) temperature*[1] *of Ann Arbor (MI, USA) on Aug. 18 (a summer day) was* $58^\circ F$ *degrees, whereas its temperature in Mar. 15 (a winter day) was* $66^\circ F$.

The end points considered for a trendline may not be readily available as data points. Instead, they may need to be aggregated over a period, which could itself be cherry-picked, as we see in the next example. In fact, even in the example above, the average daily temperature may not so neatly have been materialized, leaving us to average individual temperature readings, taken ever minutes, over a period of our choice.

EXAMPLE 2 (GIULIANI'S ADOPTION CLAIM). *Wu et al. [3] provide an example claim made by Rudy Giuliani in the 2007 Republican presidential candidates' debate that "adoptions went up 65 to 70 percent" in New York City "when he was the mayor". The claim considered the total number of adoptions during 1996-2001 v.s. 1990-1995, while Giuliani actually was in office in 1994-2001.*

In this paper, we focus on trends derived by comparing a pair of points in data to make a statement. As we can see from the examples above, such statements are quite common. [2] There are other, potentially more robust, methods to find a trendline, such as line fitting with least squared error. Our focus is restricted to trendlines based on selected endpoints. The question we ask is whether the chosen points gives us a "reasonable" trend.

Note that the trendlines are technically "not fake" in that they are derived from actual data points: our task is to determine whether the data points were cherry-picked, whether intentionally or by accident. To this end, we ask what other data points could have been chosen for a similar analysis. We can then look at the trendlines obtained from all such alternative pairs of points. The trend observed over these alternatives should not differ by much from the reported trend if it is stable. In contrast, a trendline is presumed to be cherry-picked, if it differs greatly from most alternatives considered. Even if it is not intentionally chosen to mislead, there is no question that it does mislead its consumers about the observed trend.

[1] based on www.wunderground.com
[2] A detailed discussion about the recent real-life cherry-picking instances, if those fit to the scope of this paper, and required adjustments is provided in § 9.

Of course, this begs the question of what alternatives to consider for the start and end points of a trend line. In the simplest, and most common case, there are no additional constraints, and we can define a region around the original start and end points. However, in other problems, we may have constraints. For example, we may need the distance between begin and end points to be exactly one year, or we may need the start point always to be a Monday. We formally define a rich set of constraints, covering most examples we have seen in practice.

The set of possible choices can be very large for each end-point. Therefore, the number of pairs to consider can become impractical, as in a full cartesian product. We develop efficient algorithms for computing the support of statements. For unconstrained trendlines, we design an exact linearithmic algorithm. Similarly, for constrained trendlines, we design the proper data structure and an exact linearithmic algorithm. Next, we use Monte-Carlo methods and design sampling-based approximation algorithms for computing support in scale. Providing a negative result, we also propose a randomized heuristic, effective in practice.

If we believe that a particular statement is cherry-picked, we may ask what would be an alternative statement with greater support. Towards this end, we formalize the problem of finding the trendline statement with the most support in a dataset, We present two formal definitions, and design algorithms for both. In summary, we make the following major contributions in this paper:

- We define the notion of *support* as a metric to identify cherry-picking in trendlines reported (§ 2).
- We develop efficient exact algorithms to compute support for different types of trendlines (§ 3 and § 4).
- We develop randomized algorithms based on Monte-Carlo methods to compute support, efficiently and effectively (§ 5).
- We define the notion of a most supported statement and develop algorithms for discovering it(§ 6).
- We conduct extensive experiments on real-world datasets to study the efficiency and effectiveness of our proposal, as well as providing a proof of concept (§ 7).

2. PRELIMINARIES

2.1 Data Model

In this paper, we consider the trend statements derived by comparing a pair of points. Consider a dataset $\mathcal{D}$ defined over trend attributes $\langle x_1, x_2, \cdots, x_d \rangle$ and a target attribute y which is the "measure of goodness" for each combination of target values. We use n to refer to the number of tuples in $\mathcal{D}$. The trend attributes can be continuous or discrete. For instance, consider Example 1. The trendline attributes are time and location. Location, for example, in the form of different countries is discrete, whereas the location in the form of longitude and latitude is continuous. In Example 1, the target attribute is temperature. We use a vector $\langle p[1], p[2], \cdots, p[d] \rangle$ to refer to a point p in the space value combinations for the trend attributes. For every such point p, the notation $y(p)$ is used to refer to the value of y for p. For instance, in Example 1, the vector $p = \langle$ July 20 2012, Ann Arbor - MI $\rangle$ and $y(p) = 70$ show that the temperature of Ann Arbor - MI in July 20 2012 was 70 degrees.

Typically, the dataset $\mathcal{D}$ has tuples that comprise both trend and target attributes. Therefore, given the combination of trend values, the target value can simply be looked up, for example, by using an index into the dataset. We allow more general cases where the target attribute value is somehow determined as a function of the trend values: this could be additional data collection, a function computation, or something else. We model this as an *oracle* that, given the combination of the trend values returns the target value for some "cost". The oracle cost for the traditional dataset model could be as low as $O(1)$, but in some other situations we may need to address costly oracles.

2.2 Trendline Statement

We consider trendlines based on selecting a pair of points in the space of the trend attributes. For example, in Example 1, the trendline compares the temperature of Ann Arbor on two different days. Formally, such a trendline is defined as follows:

DEFINITION 1 (TRENDLINE). *For a dataset $\mathcal{D}$, a trendline θ is a defined as a pair of trend points b (the beginning) and e (the end) and their target values in the form of $\theta = \langle (b, y(b)), (e, y(e)) \rangle$.*

Trendlines can also be made based on an *aggregate* over the target values in a window. For instance, consider Example 2. In this example, year is the trend attribute. For every year p, the target attribute $y(p)$ shows the total number of adoptions in that year. Giuliani's claim is based on the aggregate over a 5-year window. The aggregate window is identified by its beginning and its width. For example, the aggregate window 1996-2001 in Example 2 is identified by the point 1996 and 6 years as the beginning and the length of the window. Considering a fixed length for the aggregate window, for every point p, let $Y(p)$ be the aggregate over the values of $y(p')$ for all points in the window of p. Using this model, Definition 1 gets generalized for aggregate windows by replacing $y(b)$ and $y(e)$ by $Y(b)$ and $Y(e)$.

The cases for which the length of the aggregate window is not fixed can also be modeled by Definition 1, by assigning each window to its beginning point and adding an extra trend attribute showing the length of the window. For instance, in Example 2, the adoption numbers for the years 1996 to 2002 are (approximately) 3600, 4000, 3800, 3750, 3100, 2700, and 2650 [3]. The window 1996-2001 is modeled as the point $p_1 = \langle 1996, 6 \rangle$ and its target value (sum) is $Y(p_1) = 20950$. Similarly, the point $p_2 = \langle 1998, 5 \rangle$ shows a window of length 5 years that starts at year 1998; computing the sum for this window, $Y(p_2) = 16000$. Note that preprocessing the data, for the non-holistic aggregate functions such as sum and mean, while maintaining the moving aggregate, needs a single scan. Hence for a dataset of n points, it is in $O(n)$.

Following the above discussion, in the rest of the paper, we consider Definition 1 for trendlines for both single value (Example 1) and aggregate window (Example 2) trendlines.

Next, we define the trendline statements, or simply statements, as the claims that are made based on the choice of a trendline.

DEFINITION 2 (STATEMENT). *Given a trendline $\theta = \langle (b, y(b)), (e, y(e)) \rangle$, a statement is made by proposing a condition that is satisfied by the target values $y(b))$ and $y(e)$. In this paper, we consider the conditions that are made based on the absolute difference between $y(b)$ and $y(e)$. Formally, given the trendline θ, the statement S_θ is a range $(\perp, \top)$ such that:*

$$y(e) - y(b) \in (\perp, \top)$$

For instance, in Example 1 the beginning point of the trendline θ is $b = \langle$ Aug. 18 2012, Ann Arbor - MI $\rangle$ with $y(b) = 58°$F, and its end is $e = \langle$ March 15 2012, Ann Arbor - MI $\rangle$ with $y(e) = 66°$F. The statement S_θ:"summer was colder than winter" is made by proposing a condition: $(0, \inf)$, that is satisfied by θ as $y(e) -$

$y(b) = 66 - 58 > 0$. As another example, consider Example 2. Assuming a fixed window size of 6 years, the trendline θ' consists of the beginning point $b' = 1996$ and $y(b') = 0.65$ (the adoption rate for a window of 6 years starting at $b = 1996$ is 65%) and the end point $e' = 1996$ and $y(e') = 0.7$. The statement $S_{\theta'}$:"adoption rate went up" is made by proposing a condition: $(0, \inf)$, that is satisfied by θ', as $y(e') - y(b') = 0.7 - 0.65 > 0$.

2.3 Support Model

Given a statement based on a trendline, our task is to determine whether the trendline is based on cherry-picked data points. For this purpose, we consider the entire dataset and compute a "support" measure for the given statement in the dataset. We would like trendline statements to be well-supported by the data.

So, how should support be defined? Our intuition is that cherry-picked trendlines are carefully selected and, therefore, may change by slightly changing the trend points. For instance in Example 1, it turns out that perturbing the beginning and/or the end points of the chosen dates by even a few days results in trendlines for which a summer day is not colder than a winter day, i.e., $y(e) - y(b) \leq 0$. Hence, those trendlines do not support the statement that "summer was colder than winter".

Following the above discussion, given a Statement $S = (\bot, \top)$ and a trendline $\theta' = \langle (b', y(b')), (e', y(e')) \rangle$, we say that θ' supports S_θ, if $y(e') - y(b') \in (\bot, \top)$.

We name the space of trend points in which the support of a statement is studied as the *support region*. In the following, we provide two possible ways of identifying a region:
(i) Rectangular region: A rectangular region in the space of trend attributes is defined as a vector of d ranges $R = \langle [R_\vdash[1], R_\dashv[1]], \cdots, [R_\vdash[d], R_\dashv[d]] \rangle$ such that $\forall i \in [1, d] : p[i] \in [R_\vdash[i], R_\dashv[i]]$.
(ii) Circular region: A circular region in the space of trend attributes is identified by a vector $\rho = \langle \rho_1, \rho_2, \cdots, \rho_d \rangle$ and a value r. It specifies the set of points that have a maximum distance of r from the vector ρ. In fact, a rectangular support region can be regarded as a special case of a circular region, with scaled L^∞-norm.

Given a statement S, a support region for S, $R_S = (R(b), R(e))$, is defined as a pair of *disjoint* regions, where every trendline θ_i with the beginning and end points b_i and e_i should satisfy the conditions $b_i \in R(b)$ and $e_i \in R(e)$ in order to be considered for computing the support of S.

A support region may naturally be defined by the statement itself. For instance, in Example 1, the statement is made on the temperature of summer versus winter days. This naturally sets the support region to the set of summer days as $R(b)$, the valid beginnings for the trendlines, and the set of the winter days as $R(e)$, the set of end points for them. When the support region of some statement is not obvious from the statement itself, an appropriate region can be defined by a domain expert. For instance, in Example 2 it is not immediately clear if the width of the aggregation window is fixed at 6, or if an aggregation window of other sizes should also be considered? Also, it is not clear what are the valid years (for defining the aggregation windows) for the beginning and end of the trendlines. However, these questions are not difficult to answer for someone with domain knowledge.

Thus far, we have considered *unconstrained* support regions. That is every trendline $\theta = \langle (b, y(b)), (e, y(e)) \rangle$ where $b \in R(B)$ and $e \in R(E)$ is "valid" for studying the support of a statement. We name such trendlines as ***unconstrained trendlines***. However, for some statements, all possible trendlines drawn in the support region may not be valid. For instance, consider Example 1. A trend point is a combination of a location and a date/time. However, a trendline that compares the temperature of two different locations on different days does not make sense and sense is not valid. Such invalid choices must be eliminated from the support region when computing the support for a statement. We do so by formally specifying *validity constraints*.

The two extreme cases based on the validity constraints are:

1. *no-constraint*: where every trendline $\theta_i = \langle (b, y(b)), (e, y(e)) \rangle$ is valid, as long as its beginning and end points belong to the support region – i.e., $b \in R(b)$ and $e \in R(e)$.

2. *single-point enforcement*: the choice of the beginning point, enforces the end point to a single point. The trendlines that are supposed to have a fixed distance between their beginning and end points fall in this category.

We name the trendlines that require satisfying validity constraints to be considered for a given statement, as ***constrained trendlines***. Based on how restrictive the validity constraints are, the choice of the beginning of the constrained trendline limits the end points for the valid trendlines. We assume the validity constraints are provided by the expert. Still, a realistic assumption is that for all start points, the valid regions in $R(e)$ create a fixed-size window with a fixed distance from the start points. This, follows a generalization of the single-point enforcement case, that instead of having a fixed distance between the beginning and end point, we allow a *range* of distances. For instance, in our Example 1, it is like allowing the comparison in the temperature of the cities, as long as their distance is within a bounded range; or in Example 2, it is like allowing the comparison between the adoption rate of the years, so long as their differences are at least 4 and at most 6 years.

We define the support of a statement as the proportion of (valid) trendlines in $R(b)$ and $R(e)$ for which their target value difference remains within the acceptable range. Formally:

DEFINITION 3 (SUPPORT FOR A STATEMENT). *Given a data set $\mathcal{D}$, a statement $S = (\top, \bot)$, and a support region $R_S = (R(b), R(e))$, the support for S is*

$$\omega(S, R_S, \mathcal{D}) = \frac{vol(\{\text{valid } \langle p \in R(b), p' \in R(e) \rangle \mid y(p') - y(p) \in (\bot, \top)\})}{vol(\{\text{valid } \langle p, p' \rangle \mid p \in R(b), p' \in R(e)\})}$$

The denominator in Definition 3 is the universe of possible valid trendlines from $R(b)$ and $R(e)$. For unconstrained trendlines, this is the product of the "*volume*" of $R(b)$ and that of $R(e)$:

$$\begin{aligned} vol(\{\langle p, p' \rangle \mid p \in R(b), p' \in R(e)\}) &= \int_{R(b)} \int_{R(e)} dx_e dx_b \\ &= \int_{R(b)} dx \int_{R(e)} dx \\ &= vol(R(b)) \times vol(R(e)) \end{aligned} \quad (1)$$

Having the terms formally defined, next we shall formulate the problems we address in this paper.

2.4 Problem Formulation

In this paper, our goal is to design a system for detecting cherry-picked trendlines. To do so, we compute the support for a statement based on Definition 3. Formally:

PROBLEM 1. *Given a dataset $\mathcal{D}$, a trendline statement S, and a support region R_S, compute $\omega(S, R_S)$.*

While the main focus of the paper is on efficiently addressing Problem 1, we also aim to find *the statement with the highest support*. We consider two formulations for the problem:

PROBLEM 2. Most Supported Statement (MSS) for a fixed range: *Given a dataset $\mathcal{D}$, a value d, and a support region R_S find the statement $S = (\perp, \perp + d)$ with the maximum support. Formally,*

$$max\ \ \omega(S(\perp, \top), R_S)$$
$$s.t.\ \ \top - \perp = d$$

Given a fixed width for the support statement range, Problem 2 aims to find the most supported statement. An orthogonal alternative is to fix the support value. Obviously any trendline supports the range $(-\infty, \infty)$. Hence, the support of the statement $S(-\infty, \infty)$ is always 1. However, this does not provide any information about the trend since its range is too wide. The tighter the range of a statement is, the more restrictive, and hence more informative, it is. Therefore, in Problem 3, our goal is to find the tightest statement with a given support value. Formally, we define the "tightest statement (TS) for a given support" problem as following:

PROBLEM 3. Tightest Statement (TS) for a given support: *Given a dataset $\mathcal{D}$, a support region R_S, and a value $0 < s \leq 1$, find the statement $S = (\perp, \top)$ such that $\mathcal{S}(S) \geq s$ and $\top - \perp$ is minimized. Formally,*

$$min\ \ \top - \perp$$
$$s.t.\ \ \omega(S(\perp, \top), R_S) \geq s$$

Considering Problem 1 as the main focus of this paper, first in § 3 we provide an efficient exact solution for Problem 1, for unconstrained trendlines. We will introduce constraints in § 4. Later on, in § 5 we propose sampling-based approaches for approximating the support. We will study Problems 2 and 3 in § 6.

3. UNCONSTRAINED TRENDLINES

In this section we aim to design an efficient exact algorithm for the computation of the support of unconstrained trendline statement. In the following, we first propose a baseline algorithm that leads to the design of our efficient algorithm in § 3.2.

3.1 Baseline algorithm

First, let us take a careful look at Definition 3, especially the numerator of the equation. The numerator can be rewritten as a conditional integral as follows:

$$vol(\{\langle p \in R(b), p' \in R(e)\rangle \mid y(p') - y(p) \in (\perp, \top)\})$$
$$= \int_{R(b)} \Big(\int_{\{dx \in R(e) \mid y(dx_e) - y(dx_b) \in (\perp, \top)\}} dx_e \Big) dx_b \quad (2)$$

Consider the partitioning of the space into the Riemann pieces (the data records in the dataset $\mathcal{D}$). For a trend point dx_b, let $R_{dx_b}(e)$ be the points in $R(e)$ where $y(dx_e) - y(dx_b) \in (\perp, \top)$. Then, Equation 2 can be rewritten as the following Riemann sum:

$$\sum_{\forall dx_b \in R(b)} dx_b \Big(\sum_{\forall dx_e \in R_{dx_b}(e)} dx_e \Big) \quad (3)$$

Consider the example in Figure 1. The horizontal axis shows the trend attribute x while the vertical axis shows y. The trendline of interest is specified by the vertical dashed lines; the left green region identify $R(b)$ while the one in the right shows $R(e)$, and the curve shows the y values. In this example, the range of the statement S is (α, ∞). A point dx_b in $R(b)$ is highlighted in red in the left of the figure. For dx_b, all points $dx_e \in R(e)$ for which $y(dx_e) - y(dx_b) > \alpha$ support S, forming $R_{dx_b}(e)$ (highlighted in red in the right-hand side of the figure), and therefore, are counted for dx_b. The summation of these counts for all points in $R(b)$ computes the numerator of Equation 3. Following this,

Algorithm 1 BASELINE
Input: statement $S = (\perp, \top)$, support reg $R_S = \langle R(b), R(e)\rangle$
Output: $\omega(S, R_S)$

```
1: cnt = 0
2: for dx_b in R(b) do
3:    for dx_e in R(e) do
4:       if y(dx_e) − y(dx_b) ∈ (⊥, ⊤) then cnt = cnt + 1
5:    end for
6: end for
7: return cnt / (vol(R(b)) × vol(R(e)))
```

the baseline solution (Algorithm 1), sweeps a vertical line from left to right through $R(b)$ and counts the acceptable points in$R(e)$ for each dx_b (similar to highlighted dx_b and R_{dx_b} in Figure 1).

Clearly, comparing each pair of points in $R(b)$ and $R(e)$, Algorithm 1 is quadratic in the number of items in dataset, i.e. $O(n^2)$.

Next, we show how an observation about $R(b)$ and $R(e)$ lead to the design of a linearithmic algorithm.

3.2 Efficient exact algorithm

For each point $dx[i]$ in $R(b)$, the baseline algorithm makes a pass over $R(e)$ to find the set of points that, together with $dx[i]$, support the statement S, and therefore, is quadratic. In this section, we seek to design an algorithm that passes through $R(b)$ and $R(e)$ independently.

Consider Equation 3 once again. For a point $dx[i]$ in $R(b)$, let $w[i]$ be the number of points in $R(e)$ where $y(dx_e) - y(dx[i]) \in (\perp, \top)$, i.e. $\sum_{\forall dx_e \in R_{dx[i]}(e)} dx_e$. Then, Equation 3 can be rewritten as:

$$\sum_{\forall dx[i] \in R(b)} w[i] \quad (4)$$

For example, in Figure 1, the weight of the point dx_b is the width of the red rectangle $R_{dx_b}(e)$. In the following, we show how the construction of a cumulative function for $R(e)$ enables efficiently finding the corresponding weights for the points in $R(b)$.

Let us consider the example of Figure 1 once again. Let $dx[1]$ to $dx[n']$ be the set of points in $R(b)$, from left to right. Figure 2 shows three points $dx[i]$, $dx[j]$, and $dx[k]$ where $y(dx[i]) < y(dx[j]) < y(dx[k])$. It also highlights $R_{dx[i]}(e)$, $R_{dx[j]}(e)$, and $R_{dx[k]}(e)$ in the right. Note that $R_{dx[i]}(e)$ consists of two disjoint rectangles. Looking at the figure, one can confirm that $R_{dx[k]}(e)$ is a subset of $R_{dx[j]}(e)$ and $R_{dx[j]}(e)$ is a subset of $R_{dx[i]}(e)$. Since all points in $R_{dx[k]}(e)$ belong to $R_{dx[j]}(e)$ and $R_{dx[i]}(e)$, we don't need to recount those points three time for $dx[i]$, $dx[j]$, and $dx[k]$. Instead, we could start from $dx[k]$, compute its width, move to $dx[j]$, only consider the parts of $R_{dx[j]}(e)$ that is not covered by $R_{dx[k]}(e)$, i.e. $R_{dx[k]}(e) \backslash R_{dx[k]}(e)$, and set $w[j]$ as $w[i]$ plus the width of the uncovered regions by $R_{dx[k]}(e)$. Similarly, in an incremental manner, we could compute $w[i]$, as we sweep over $R(e)$.

Following the above discussion, if we could design a "cumulative" function $F : \mathbb{R} \rightarrow \mathbb{R}$, that for every value y, returns the number of points dx in $R(e)$ where $y(dx) < y$, we could use it to directly compute the weights for the points in $R(b)$. Formally, we seek to design the following function F:

$$F(y) = |\{dx \in R(e) \mid y(dx) < y\}| \quad (5)$$

Given such a function F, the weight of the point $dx[i] \in R(b)$ can be computed as following:

$$w[i] = F\big(y(dx[i]) + \top\big) - F\big(y(dx[i]) + \perp\big) \quad (6)$$

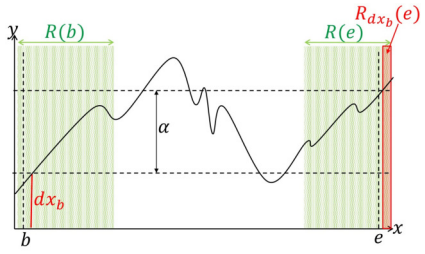

Figure 1: Illustration of a point dx_b and the set of points in $R(e)$ for which $y(dx_e) - y(dx_b) \geq \alpha$.

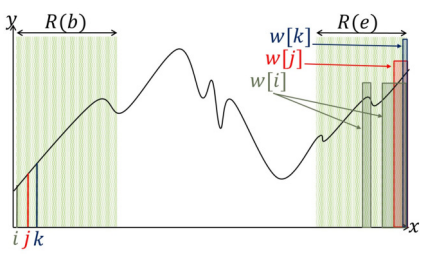

Figure 2: Illustration of weights for three points $dx[i]$, $dx[j]$, and $dx[k]$ in the example of Figure 1.

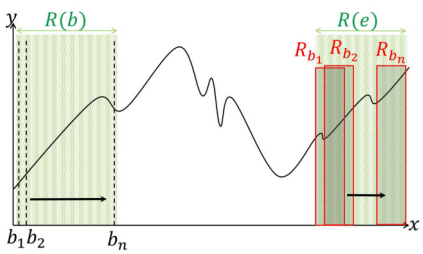

Figure 3: Illustration of the sliding window in $R(b)$ for constrained trendlines.

Algorithm 2 EXACT$_u$
Input: statement $S = (\perp, \top)$, support reg. $R_S = \langle R(b), R(e) \rangle$
Output: $\omega(S, R_S)$

1: $\mathfrak{F} = [\,]$
2: **for** dx in $R(e)$ **do** add $y(dx)$ to $\mathfrak{F}$
3: sort $\mathfrak{F}$ descending ; $cnt = 0$
4: **for** dx in $R(b)$ **do**
5: $\quad i_1 = \textit{binary\_search}(y(dx) + \perp, \mathfrak{F})$
6: $\quad i_2 = \textit{binary\_search}(y(dx) + \top, \mathfrak{F})$
7: $\quad cnt = cnt + (i_2 - i_1)$
8: **end for**
9: **return** $\frac{cnt}{vol(R(b)) \times vol(R(e))}$

But first, we need to implement the function F. We use a sorted list $\mathfrak{F}$ for the implementation of F. $\mathfrak{F}$ contains the target values in $R(e)$ such that the i-th element in $\mathfrak{F}$ shows the y value for the i-th largest point in $R(e)$.

Having the target values sorted in $\mathfrak{F}$, in order to find $F(y)$, it is enough to find index i for which $\mathfrak{F}[i] < y$ and $\mathfrak{F}[i+1] \geq y$. Then, $F(y) = i$. That is because, for all $j \leq i$: $\mathfrak{F}[j] < y$, while for all $j > i$: $\mathfrak{F}[j] \geq y$. Therefore, the number of points for which $y(x) < y$ is equal to i. Also, since the values in $\mathfrak{F}$ are sorted, we can use binary search for finding the index i.

In order to compute the weight of a point point $dx \in R(b)$ using the Equation 6, Algorithm 2 conducts two binary searches on $\mathfrak{F}$. It uses the sums of the weights and calculate $\omega(S, R_S)$.

As explained above, the EXACT$_u$ algorithm (Algorithm 2) has two phases: (i) constructing the sorted list $\mathfrak{F}$, and (ii) parsing over the points in $R(b)$ and calculating the support. (i) is in $O(n \log n)$. In (ii), for each point in $R(b)$, the algorithm runs two binary searches over the array F (of size n) and, therefore, is again in $O(n \log n)$. Hence, the EXACT$_u$ algorithm is in $O(n \log n)$.

Having designed the efficient algorithm for unconstrained trendlines, next we extend it to constrained trendlines in § 4.

4. CONSTRAINED TRENDLINES

So far, our focus was on the unconstrained trendlines, where $\theta = \langle (b, y(b)), (e, y(e)) \rangle$ is valid, so long as its beginning and end points belong to the support region – i.e., $b \in R(b)$ and $e \in R(e)$.

In this section, we consider computing the support of the statements that are based on constrained trendlines. Recall that the single-point enforcement is the extreme case of constrained trendlines where the choice of the beginning point, enforces the end point to a single point. Computing the support for these cases is straightforward. To do so, given a support statement $S = (\perp, \top)$ over the support region $R_S = (R(b), R(e))$, it is enough to make a pass over the points in $R(b)$, for each point $b \in R(b)$ find its corresponding point $e \in R(e)$, and count up if $y(e) - y(b) \in (\perp, \top)$. This simple algorithm is linear in the size of $R(b)$.

A similar approach also works for the less extreme constrained trendlines. For a point $b \in R(b)$, let $R_b(e)$ be the set of valid points in $R(e)$. One can make a pass over $R(b)$, and for each point $b \in R(b)$ find $R_b(e)$. Then, it is enough to, for each $b \in R(b)$, count the number of trendlines that support S. This algorithm, apparently, is efficient when the size of $R_b(e)$ is small. Especially, when $R_b(e)$ is a small constant, just like the single-point enforcement case, the algorithm is linear to the size of $R(b)$. The problem is, however, when $R_b(e)$ is a considerably large portion of $R(e)$. For example, for the constrained trendlines with more freedom where $|R_b(e)|$ is in $O(n)$, the algorithm becomes quadratic, i.e., $O(n^2)$ – just like our baseline in § 3.1.

Our aim in this section is to maintain the linearithmic performance. We note that if $|R_b(e)| \leq O(\log n)$, the baseline solution is in $O(n \log n)$; that is, it already is efficient.

In § 3.2, we proposed the construction of the cumulative function F. The cumulative function gave us the advantage to, for every point $b \in R(b)$, find the number of points in $e \in R(e)$ that $\theta = \langle b, e \rangle$ supports the statement S. Recall that we use a sorted list, containing the objective values of the points in $R(e)$, for the development of F. Then, every call of F is equivalent with conducting a binary search on the list, and, hence, is in $O(\log n)$. This method becomes problematic for the constrained trendlines, as not all the elements in F correspond to a valid point $R_b(e)$. One still could make a pass over F and filter out the invalid points, and then apply the binary search. However, requiring a pass over F, this reduces the performance of F to $O(n)$, dropping the overall performance of the algorithm to $O(n^2)$.

The other alternative is to consider $R_b(e)$ as a *sliding window* while sweeping over $R(b)$. That is, to initially find $R_b(e)$ for a (corner) point in $R(b)$, and to move $R_b(e)$, as a sliding window, while sweeping b (Figure 3). Then, initially constructing the sorted list of objective values for the first region $R_b(e)$, while sliding the window, one can update the list by removing the points that are no further valid, and adding the new points in $R(e)$ that become valid.

Maintaining the objective values of the points inside the window, however, is problematic when there are updates. That is because every insert or delete into the list requires to shift the values in the array which makes the performance of each count linear in $|R_b(e)|$, even though the binary search is still in $O(\log n)$. Alternatively, one could use a heap data structure for maintaining the sorted list. However, even though the operations are efficient in heap, it is not possible to conduct a binary search of $O(\log n)$ on it. This again, makes the final algorithm quadratic.

In order to develop the sliding window strategy, we need to be able to update (both insert and delete) and also the search in a logarithmic time. The quick answer is a balanced binary search tree. Red-black trees (RBT) [4], is balanced binary search tree (BST) that has a logarithmic run-time for insert, delete, and also search. But there still is a small issue. Search in a BST checks the existence

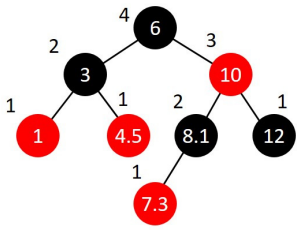

Figure 4: A sample red black tree in which every node contains the size of its left sub tree (including the node itself).

of an item in the list and if not a pointer to the location it can get inserted. We, instead need to *count* the number of elements smaller than the queried value in the tree. Given a BST, one would need to iterate over its nodes in the "left-hand side" of the key, in order to find the number of nodes with smaller value.

In the following, we show how we adapt BST for finding the counts efficiently. The central idea is that if the nodes of the BST, in addition to the node value, maintain the size of their left sub-tree, then counting the number of nodes in the left-hand side of the key is doable in logarithmic time. Consider the sample RBT in Figure 4. Note that, in addition to the node values, every node contains the count of its left sub tree (including the node itself). In order to count the number of nodes smaller than a given key value, starting from the root and following the path for finding the key, it is enough to add up the count values for the nodes with smaller values than the key. For instance, in Figure 4, let the key be 8, that is, the goal is to compute $F(8)$. Let *sum* (initially 0) be the variable containing the sum of counts. The traverse starts from the root and since its value (6) is smaller than the key (8), its count value is added to *sum* making its value 4. It then moves to the right child of the root with the value of 10. Since $10 > 8$, it then moves to its left child (8.1) and since that also is larger than 8, moves to its left child with value 7. Then, because $7 < 8$, it adds its count value to *sum* making it $4 + 1 = 5$. The algorithm then stops since this node does not have a right child, returning $F(8)$ as 5.

The only remaining issue here is updating the counts while updating the red-black tree. The RBT does not originally contain the count, and does not consider it while updating the tree. Similar to the search operation, insertion and deletion operations, traverse the RBT from the root, while at each iteration conducts constant number of "rotation" operations (for further details please refer to [4]). Rotation is a *local operation* changing $O(1)$ pointers in the tree. Every rotation operation may change the structure of the tree by moving the *entire* sub-tree under a node to another node. Therefore, assuming that every node maintains the the size of its left sub-tree (including itself), updating the counts for on each rotation is also in $O(1)$ – without the need to traverse over the sub-trees. Consequently, adding the count values to the nodes of RBT does not affect its logarithmic update time.

Having the proper data structure for developing the sliding window strategy, Algorithm 3 shows the pseudocode of EXACT$_c$, the efficient algorithm for computing the support of statement with constrained trendlines. If the constraints on the validity of trendlines are restrictive enough that makes the size of $R_b(e)$ less than $\log n$, then we follow the baseline strategy for computing the support, which gives a performance of $O(n \log n)$. Otherwise, the algorithm uses the RBT (with count values) data structure, illustrated in Figure 4. Then, starting from the left-most point in $R(b)$, it follows the sliding window strategy, keeping the RBT up-to-date, as it moves the window. Let the number updates to RBT upon moving the window be k. Then the algorithm needs to conduct $O(k)$ updates per move, making its overall run-time in $O(k\, n \log n)$.

So far in § 3 and 4, we designed efficient exact algorithms that, as we shall demonstrate in § 7, run well for the large settings. However, for the very large settings, the exact algorithms may not be very efficient. On the other hand, as an aggregate value, the user may prefer a quick, yet accurate, estimation of the support over waiting for the exact value. She therefore, may be willing to trade-off accuracy with efficiency. Following this, next in § 5, we seek to design a sampling-based algorithm for estimating the support of trendline statements.

Algorithm 3 EXACT$_c$
Input: statement $S = (\perp, \top)$ and support region $R_S = \langle R(b), R(e) \rangle$
Output: $\omega(S, R_S)$

$cnt = 0$, $vol = 0$
if size of valid region for a point $b \in R(b) \leq O(\log n)$ **then**
  for $b' \in R(b)$ **do**
    $R_{b'}(e) = \{e' \in R(e) \mid \langle b', e' \rangle$ is a valid trendline$\}$
    $vol = vol + |R_{b'}(e)|$
    for $e' \in R_{b'}(e)$ **do**
      if $y(e') - y(b') \in (\perp, \top)$ **then** $cnt = cnt + |R_{b'}(e)|$
    end for
  end for
  return $\frac{cnt}{vol}$
end if
RBT $= new$ red-black tree
$b' =$ the left-most point in $R(b)$
for $e' \in R_{b'}(e)$ **do** add(RBT, $y(e')$)
while true **do**
  $vol = vol +$ |RBT|
  $i_1 =$ *count_smaller*(RBT, $y(b') + \perp$)
  $i_1 =$ *count_smaller*(RBT, $y(b') + \top$)
  $cnt = cnt + (i_2 - i_1)$
  $b' =$ *sweep_to_next* $(b', R(b))$
  if $b' =$ null **then break**
  $X_1 =$ the points to be removed from RBT
  for $e' \in X_1$ **do** $remove$(RBT, $y(e')$)
  $X_2 =$ the points to be added to RBT
  for $e' \in X_2$ **do** add(RBT, $y(e')$)
end while
return $\frac{cnt}{vol}$

5. RANDOMIZED ALGORITHM

In very large settings where the number of points in $R(b)$ and $R(e)$ is significant, or in the absence of explicit target values where acquiring the data is costly, exact algorithms may not be efficient. On the other hand, a precise-enough estimation of the support of a statement may give a good idea of whether or not it is cherry-picked. Hence a user may prefer to quickly find the estimate, rather than spending a significant amount of time for finding out the exact values. In this section, we seek to design a Monte-Carlo method [5, 6] for estimating the support of a statement.

Monte-Carlo methods use repeated sampling and the central limit theorem [7] for solving deterministic problems. Based on the law of large numbers [7], the mean of independent random variables can serve for approximating integrals. That is because the expected number of occurrence of each observation is proportional to its probability. At a high level, the Monte-Carlo methods work as follows: first, they generate a large enough set of random inputs based on a probability distribution over a domain; then they use these inputs to estimate aggregate results.

In the following, we first, § 5.2, propose the PAIRSAMPLING algorithm that is based on sampling pairs of points from $R(b)$ to $R(e)$ and works both for unconstrained and constrained trendlines. Then,

Algorithm 4 PAIRSAMPLING
Input: statement $S = (\perp, \top)$, support region $R_S = \langle R(b), R(e) \rangle$, a dataset $\mathcal{D}$, the sampling budget N, confidence level α

$cnt = 0$
for $i = 1$ to N **do**
  $p =$ a uniform sample from $R(b)$
  $p' =$ a uniform sample from $R_p(e)$
  if $(y(p') - y(p)) \in (\perp, \top)$ **then** $cnt = cnt + 1$
end for
$m_\xi = \frac{cnt}{N}$
$e = Z(1 - \frac{\alpha}{2})\sqrt{\frac{m\ (1-m)}{N}}$
return m_ξ, e

we will provide POINTSAMPLING in § 5.2 for unconstrained trendlines that sample independent points $R(b)$ and $R(e)$. Although both PAIRSAMPLING and POINTSAMPLING provide unbiased estimations, we provide a negative theoretical result about the variance of POINTSAMPLING. Still as we shall show in in § 7, in all of our experiments, for a fixed sampling budget, POINTSAMPLING provided low variance estimations, tighter than PAIRSAMPLING. POINTSAMPLING provides better estimations when sampling is costly and the number of samples are limited.

5.1 Pair sampling

PAIRSAMPLING is a monte-carlo estimation algorithm that follows the idea of drawing independent trendline samples for estimating the support of a statement (both unconstrained and constrained). Consider the trendline statement $S = (\perp, \top)$ with the support region $R_S = \langle R(b), R(e) \rangle$. The universe of possible trendlines from $R(b)$ to $R(e)$ is the set of valid pairs $\langle p, p' \rangle$ where $p \in R(b)$ and $p' \in R(e)$. Let ω be the support of S in the region R_S, i.e., $\omega(S, R_S)$. For each uniformly sampled pair $\langle p, p' \rangle$, let the random Bernoulli variable $x_{\langle p,p' \rangle}$ be 1 if $y(p') - y(p) \in (\perp, \top)$, 0 otherwise. The probability distribution function (pdf) of the Bernoulli variable x is:

$$p(x) = \begin{cases} \omega & x = 1 \\ 1 - \omega & x = 0 \end{cases} \tag{7}$$

The mean and the variance of a Bernoulli variable with the success probability of x is $\mu = \omega$ and the variance is $\sigma^2 = \omega\ (1 - \omega)$, respectively. For every set ξ of N iid (independent and identically distributed) samples taken from the above binary variable x, let m_ξ be the random variable showing the average of ξ. Using the central limit theorem, m_ξ follows the Normal (AKA Gaussian) distribution $\mathrm{N}(\mu, \frac{\sigma}{\sqrt{N}})$ – the Normal distribution with the mean μ and standard deviation $\frac{\sigma}{\sqrt{N}}$.

Given a confidence level α, the confidence error e identifies the range $[m_\xi - e, m_\xi + e]$ where:

$$p(m_\xi - e \leq \mu \leq m_\xi + e) = 1 - \alpha \tag{8}$$

Using the Z-table:

$$e = Z(1 - \frac{\alpha}{2}) \frac{\sigma}{\sqrt{N}} \tag{9}$$

For a large enough value of N, we can estimate σ as $\sqrt{m_\xi\ (1 - m_\xi)}$. Hence, Equation 9 can be rewritten as:

$$e = Z(1 - \frac{\alpha}{2}) \sqrt{\frac{m_\xi\ (1 - m_\xi)}{N}} \tag{10}$$

Following the above discussion, the algorithm PAIRSAMPLING (Algorithm 4) uses a budget of N sample trendlines from R_S to estimate the support $\omega(S, R_S)$. The algorithm computes m_ξ by ratio of samples that support S. It then computes the confidence error e, using Equation 10 and returns m_ξ and e. It is easy to see that, since the algorithm linearly scans over N samples, it is running time is in the order of $O(N)$.

An observation is that, in order to take N trendline samples, PAIRSAMPLING takes N independent samples from $R(b)$ and also N from $R(e)$. It, however, does not reuses the sampled points, as it intends to draw independent trendline samples. Although, reusing the points would increase the number of sampled trendlines. Next, we propose POINTSAMPLING, the algorithm that reuses the sampled points to get N^2 sample trendlines.

5.2 Point sampling, a practical solution

PAIRSAMPLING works by drawing independent trendline samples; hence it does not reuse sampled points from $R(b)$ and $R(e)$. Reusing the sampled points could result in more trendline samples and would have the potential to reduce the estimation error.

Let B be a set of N iid random samples from $R(b)$ and E a set of N iid random samples from $R(e)$. The combinations of the points in B and E generate N^2 Bernoulli samples based on Equation 7 which may result in more accurate estimations. POINTSAMPLING uses this idea. It uses the these N^2 samples and returns their average as its estimation for support. In the following, we first provide an efficient development of the POINTSAMPLING algorithm. Then, we show the negative result that, even though the number of samples increase to N^2, since the sampled pairs are not independent, POINTSAMPLING may not necessarily generate more accurate results. Still, POINTSAMPLING is effective when the sampling budget is limited, or sampling is costly. As we shall show in § 7, POINTSAMPLING had a lower variance than PAIRSAMPLING in all experiments we conducted with a fixed sampling budget.

5.2.1 *Algorithm Development*

The straight-forward development would literally generate all N^2 pairs between B and E, and similar to Algorithm 4 compute the ratio of pairs for which $(y(p') - y(p)) \in (\perp, \top)$ to all pairs. This, however, is in $O(N^2)$, simply because there are N^2 pairs it iterates over.

Instead, in the following we propose the efficient implementation, similar to Algorithm 2, which is linearithmic to N, i.e., $O(N \log N)$. Note that for every sample point b in B our objective is to find the number of points e in E such that $y(e) \in (\perp + y(b), \top + y(b))$. That is the number of points in E with the objective value of less than $\top + y(b)$ minus the ones with the objective value of less than $\perp + y(b)$.

Similar to Algorithm 2, we first design the "cumulative" function $F : \mathbb{R} \to \mathbb{R}$, that for every value y returns the number of points e in E where $y(e) < y$. Given the objective values of the points in E, the sorted list of these values represent $\mathfrak{F}$. Then, finding a value $F(y)$ is possible by applying a binary search on $\mathfrak{F}$. Algorithm 5 shows the development of POINTSAMPLING using this strategy.

5.2.2 *Variance Analysis*

In the following we study the variance of POINTSAMPLING. We will show that, even though reusing the sampled points increase the number of samples to N^2, due to the dependency between the samples, the variance will not necessarily drop.

Let $b_1, \cdots, b_N$ be the set of points sampled from $R(b)$ and $e_1, \cdots, e_N$ be the ones sampled from $R(e)$. Also, let $x_{i,j}$ be the Bernoulli variable based on Equation 7, defined over the sample

Algorithm 5 POINTSAMPLING
Input: statement $S = (\perp, \top)$, support region $R_S = \langle R(b), R(e) \rangle$, a dataset $\mathcal{D}$, the sampling budget N, confidence level α

```
for i = 1 to N do
    add a uniform sample from R(b) to B
    add a uniform sample from R(e) to E
end for
F = sorted(y(E[1]) ··· y(E[N]))
cnt = 0
for i = 1 to N do
    i1 = binary_search(y(B[i]) + ⊥, F)
    i2 = binary_search(y(B[i]) + ⊤, F)
    cnt = cnt + (i2 − i1)
end for
m_ξ = cnt / N^2
return m_ξ
```

points b_i and e_j. Recall that the mean and the variance of such variable is $\mu_x = \omega$ and $\sigma_x^2 = \omega(1-\omega)$. Using these $m = N^2$ trendline samples, POINTSAMPLING outputs the averages $x_{i,j}$, $\forall i, j \in [1, N] \times [1, N]$. Using these notations, we compute the variance of the average of $x_{1,1}$ to $x_{N,N}$. That is:

$$Var[\frac{1}{m}\sum_{i=1}^{m} x_i] = \frac{1}{m^2}\big(E[(\sum_{i=1}^{m} x_i)^2] - E[\sum_{i=1}^{m} x_i]^2\big)$$
$$= \frac{1}{m^2}\big(m\,\sigma_x^2 + \sum_{i=1}^{m}\sum_{j=1\neq i}^{m} Cov(x_i, x_j)\big) \quad (11)$$

Based on Equation 11, the variance of POINTSAMPLING depends on the covariances between the sample pairs. For a sample pairs $x = x_{i,j}$ and $x' = x_{i',j'}$ where $i \neq i'$ and $j \neq j'$, $Cov(x, x') = 0$, simply because those are independent, as b_i, $b_{i'}$, e_j, and $e_{j'}$ are drawn independently.

For the pairs where $i = i'$ or $j = j'$, though, the covariance is not zero[3]. Consider the pairs $x = x_{i,j}$ and $x' = x_{i,j'}$ (the covariance for $x = x_{i,j}$ and $x' = x_{i',j}$ is also the same):

$$Cov(x, x') = E[x\,x'] - E[x]E[x']$$
$$= E[x_{i,j}x_{i,j'}] - E[x_{i,j}]E[x_{i,j'}]$$
$$= E[x_{i,j}x_{i,j'}] - \omega^2 \quad (12)$$

Using γ to show $E[x_{i,j}x_{i,j'}]$ (and $E[x_{i,j}x_{i',j}]$)[4]:

$$Cov(x, x') = \gamma - \omega^2$$

Every sample point $b_i, \forall i = 1$ to N, there are N Bernoulli variables, $x_{i,1}$ to $x_{i,N}$, that share b_i. Hence, for each i, there are $\frac{N(N-1)}{2}$ pairs $x_{i,j}$ and $x_{i,j'}$ where covariance is not zero. Similarly, every sample point $e_j, \forall j = 1$ to N, there are N Bernoulli variables, $x_{1,j}$ to $x_{N,j}$, that share e_j, giving $\frac{N(N-1)}{2}$ pairs with non-zero covariances. Excluding the pairs that share both b_i and e_j, there totally are $2N \cdot \frac{N(N-1)}{2} = N^2(N-1)$ pairs for which covariance is not zero. Therefore:

$$\sum_{i=1}^{m}\sum_{j=1\neq i}^{m} Cov(x_i, x_j) = N^2(N-1)(\gamma - \omega^2) \quad (13)$$

[3]Note that either $i \neq i'$ or $j \neq j'$, as otherwise x and x' are the same

[4]$\gamma = E[x_{i,j}x_{i,j'}]$ is the probability that for a random sample b_i from $R(b)$ and two independent random samples e_j and $e_{j'}$ from $R(e)$, both $y(e) - y(b) \in [\perp, \top]$ and $y(e) - y(b') \in [\perp, \top]$.

Consequently:

$$Var[\frac{1}{m}\sum_{i=1}^{m} x_i] = \frac{1}{m^2}\big(m\,\sigma_x^2 + N^2(N-1)(\gamma - \omega^2)\big)$$
$$= \frac{1}{N^2}\big(\,\omega(1-\omega) + (N-1)(\gamma - \omega^2)\big) \quad (14)$$

Assuming that sampling is in $O(1)$, the computation cost between PAIRSAMPLING with $N \log N$ independent pairs of points is equal to the cost of POINTSAMPLING with the budget N. Hence, for a similar computation cost, POINTSAMPLING has a better variance than PAIRSAMPLING, if:

$$\frac{1}{N^2}\big(\,\omega(1-\omega) + (N-1)(\gamma - \omega^2)\big) < \frac{\omega(1-\omega)}{N \log N} \quad (15)$$

Substituting the variables:

$$\frac{(N-1)\log N}{N - \log N} < \frac{\omega(1-\omega)}{\gamma - \omega^2} \quad (16)$$

Let $\varphi(S) = \omega(1-\omega)/(\gamma - \omega^2)$, then:

$$\frac{N-1}{N - \log N}\log N < \varphi(S) \quad (17)$$

Note that $\varphi(S)$ is a function of the input statement S. Based on Equation 17, assuming that sampling has a constant cost, for a fixed time budget, PAIRSAMPLING is preferred over POINTSAMPLING, unless the time is very limited. On cases that sampling is costly, though, as we shall show in § 7, POINTSAMPLING is a better alternative, as in all of our experiments it outperformed PAIRSAMPLING when both used N as the sampling budget.

6. MOST SUPPORTED STATEMENT

So far in this paper, we studied the verification problem: given a statement, compute its support based on the data. Suppose the statement has a low support. An immediate follow-up question one may ask is: *if not this, what is the right statement supported by the data?* For example, consider the statement of Example 1 that in 2012 the Northern Hemisphere summer days were colder than winters. In §7.2 we shall show that this statement has a very low support. After providing this information, a natural question would be: what is the statement supported by data? Answering this question is our focus in this section. That is, instead of providing blind support numbers, we provide extra information that can be viewed as explanation to the user by comparing what is supported by data v.s. what has been stated. Specifically, we aim to find the statement (with a fixed range) that has the maximum support (problems 2) and the statement (with minimum range) for a specific support (§2). For instance, using Example 1 in § 7.2, we shall find statements with support of 80% across different cities in Northern Hemisphere. Finding most supported statements is challenging. That is because a brute force solution needs to generate *all* possible statements and check the support for each using the techniques provided in the previous sections. Let y_{min} and y_{max} be $\min(\,y(R(e)))$ and $\max(y(R(e)))$ respectively. For MSS, $(y_{max} - y_{min})$ provides a lower bound for $\perp$ and $(y_{max} - y_{min} - d)$ is an upper bound for it. The brute-force algorithm can start from the lower bound, check the support of $S(\perp, \perp + d)$, increase the value of $\perp$ by a small value ϵ, check the support of the new statement, repeat this process until $\perp$ reaches the upper bound, and return the statement with the maximum support. Note that in addition to the efficiency issue, this algorithm cannot guarantee the discovery of the optimal solution, no matter how small ϵ is.

Algorithm 6 TS
Input: $R_S = \langle R(b), R(e) \rangle, \mathcal{D}$, a support value s

for $b' \in R(b)$ **do**
  for valid e' in $R_{b'}(e)$ **do** add $y(e') - y(b')$ to ℓ
end for
sort ℓ; $\delta = s \times |\ell|$, $min = \infty$
for $i = 1$ to $|\ell| - \delta$ **do**
  if $min > \ell[i+\delta] - \ell[i]$ **then**
    $min = \ell[i+\delta] - \ell[i]$; $S = (\ell[i], \ell[i+\delta])$
  end if
end for
return S, min

Algorithm 7 MSS
Input: $R_S = \langle R(b), R(e) \rangle, \mathcal{D}$, a statement range width d

for $b' \in R(b)$ **do**
  for valid e' in $R_{b'}(e)$ **do** add $y(e') - y(b')$ to ℓ
end for
sort ℓ; $max = 0$
for $i = 1$ to $|\ell|$ **do**
  j =b-search(ℓ, i,$|\ell|$,$key = \ell[i] + d$)
  if $j = -1$ `/*not found*/` **then break**
  if $max < (j - i)$ **then**
    $max = (j - i)$; $S = (\ell[i], \ell[j])$
  end if
end for
return $S, max/|\ell|$

Instead, we first create the "sorted distribution of trendlines". That is, we create a sorted list ℓ (from smallest to largest) where every value is the difference between the target values of a valid trendline. We shall show that it enables answering both MSS and TS. Constructing ℓ requires passing over the pairs of trendlines and then sorting them. Given that the number of pairs is in $O(n^2)$, constructing the ordered list is in $O(n^2 \log n)$.

Finding the tightest statement for a given support value, using ℓ, requires a single pass over the list. Algorithm 6 shows the pseudo-code for finding TS. Starting from the beginning of the list, the algorithm moves a window of size $\delta = |\ell| \times s$ over the list. Note that the window i (the window at step i) identifies a set of δ trendlines having the indices from i to $i + \delta$ in the sorted list. Also, all trendlines in the list support the statement $S = (\ell[i], \ell[i + \delta])$. Moreover, since $s = \delta/|\ell|$, $\omega(S(\ell[i], \ell[i + \delta])) \geq s$. Using this observation, while moving the sliding window over ℓ, at every step the algorithm measures $\ell[i+\delta] - \ell[i]$ as the tightness of the current statement. The algorithm keeps track of the tightest statement and if $\ell[i+\delta] - \ell[i]$ is smaller than this value, it updates the tightest statement to i. After sweeping the window over ℓ, the algorithm returns TS. Obviously sweeping the window over ℓ is in $O(|\ell|) = O(n^2)$. Therefore the total running time of the algorithm is determined by the construction of ℓ, i.e., $O(n^2 \log n)$.

Finding MSS using ℓ is also possible, applying a similar process. The idea for MSS is also to slide a window over ℓ. The difference, however, is that the window size is not fixed anymore. Recall that every value in ℓ represents the target-value difference of a valid trendline. For a fixed statement range, the support window should contain all trendlines that their target-value differences belong to the statement range; hence, the window size is variable. The algorithm for finding MSS is provided in Algorithm 7. Starting from the beginning of ℓ the algorithm sweeps a window over ℓ. In every step i, it applies a binary search over ℓ to find the index j, such that $(\ell[j] - \ell[i]) \leq d$ while $(\ell[j+1] - \ell[i]) > d$. The support of the statement identified by the current window is $(j - i)/|\ell|$. In the end, the window with the maximum size (therefore maximum support) is returned. Applying a binary search at every step, this algorithm is in $O(n^2 \log n)$.

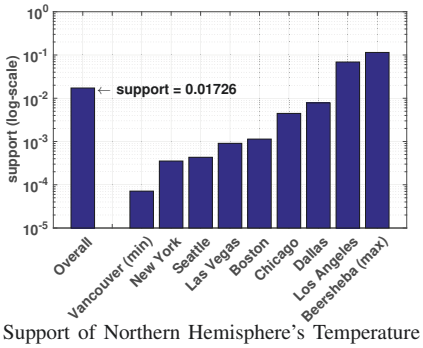

Figure 5: Support of Northern Hemisphere's Temperature Statement in 2012 (Example 1) for different cities

7. EXPERIMENTS

7.1 Experiments setup

The experiments were conducted using a 3.8 GHz Intel Xeon processor, 64 GB memory, running Linux. The algorithms were implemented using Python 2.7.
We use the following real datasets in our experiments:

I. *Carbon Dioxide Levels (CO2) [8]:* Using data from Earth System Research Lab [9], this dataset contains Atmospheric Carbon Dioxide Dry Air Mole Fractions from quasi-continuous daily measurements at Mauna Loa, Hawaii. It contains 19,127 records from Mar. 1958 to Feb. 2019, over the attributes `date` and `price`.

II. *Weather dataset (WD) [10]:* Collected from the OpenWeatherMap website[5], it contains the historical weather data between the years 2012-2017 over various weather attributes, for 36 cities in US, Canada, and Israel. For each city, the dataset contains 42,253 records, each containing hourly weather measurements such as `temperature`, `humidity`, and `air pressure`.

III. *US Department of Transportation flight dataset (DOT) [11]:* This dataset is widely used by third-party websites to identify the on-time performance of flights, routes, airports, and airlines. After removing the records with missing values, the dataset contains 457,892 records, for all flights conducted by the 14 US carriers in the last month of 2017. Each record contains measurements such as `Air-time`, `Distance`, and `Arrival-Delay`.

IV. *Stock dataset [12]:* This is our very large dataset. It contains 21 million records about the daily stock prices for a selection of several thousand stock tickers from NYSE and NASDAQ. Every tuple contains 8 attributes `ticker, open, close, adj_close, low, high, volume,` and `date`.

In the following, first in § 7.2, we demonstrate a proof of concept by studying Example 1, using the weather dataset. Then in § 7.3, we conduct the performance evaluation of our proposal.

7.2 Proof of concept – Northern Hemisphere's Temperature

In this section, we demonstrate the validity of our proposal by studying the claim in Example 1 that summer in the northern hemisphere was colder than winter in 2012-13. We use the weather dataset: $\langle x_1$: date/time , x_2 : city_name$\rangle$, y : temperature. We consider the comparison between a summer day and a winter day

[5] `openweathermap.org`

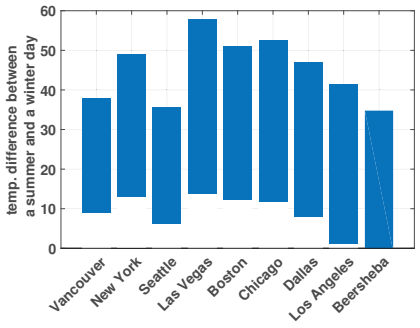

Figure 6: Tightest statements with support 0.8 for the temperature of different cities in 2012 (Example 1)

to be valid if it is comparing the temperature of the same city. Therefore, the support regions R_S is naturally defined as set of winter days of cities as the beginning (R_B) and set of summer days of cities as the end (R_E). Following this, we define R_S as $R_S = (R_B, R_E)$, where $R_B = (\langle$ Dec. 1 2012, $x_2\rangle, \langle$ Mar. 1 2013, $x_2\rangle)$ and $R_E = (\langle$ Jun. 1 2013, $x_2\rangle, \langle$ Sep. 1 2013, $x_2\rangle)$. Given R_S, the statement S is defined as $S = (-\infty, 0)$. Figure 5 shows the overall support, the cities with minimum and maximum support of the statement, as well as the support for 7 major US cities. Vancouver with a support of less than 0.0001 had the minimum support, while Beersheba with 0.1 had the maximum support. The overall support of this statement was *less than 0.018*. This means that among all valid trendlines, less than 0.018 of them support the idea that summer was colder than winter! This very small support clearly shows that the trendline based on which the statement is made is cherry-picked.

Next, to find "fair" statements supported by data, we run Algorithm 6 to find tightest statements with support $\omega = 0.9$ for different cities. Figure 6 show the results. The results confirm that summer days have been warmer than winter. Among these, the maximum difference belonged to Las Vegas where summer days where 14 to 58 degrees warmer than the winter, while the least belonged to Beersheba where the differences between summer and winter days where between -0.6 and 34 degrees.

7.3 Performance evaluation

Having demonstrated a proof of concept, in the rest of the section we evaluate the efficiency and effectiveness of our proposed algorithms. In addition to baseline (Algorithm 1), we compare our results against [3] (labeled as CFCQP). In the plots, we label our efficient exact algorithms (Algorithm 2 and Algorithm 3) as EXACT. Our default randomized algorithm is PAIRSAMPLING (Algorithm 4) which is labeled as RANDOMIZED in the plots. Since our objective in this section is to study efficiency, as the default setting, we partition the data in two equal-sized partitions $R(b)$ and $R(e)$. The following are our experiment results.

7.3.1 Unconstrained Trendlines

Following the structure of the paper, first we study unconstrained trendlines. To do so, we run [3] (CFCQP), Algorithms 1 (Baseline), and Algorithm 2 (EXACT$_u$) on different datasets while varying the input size n. The results are provided in Figure 7, Figure 8, Figure 9, and Figure 10. We note that adopting [3] for computing the support of trendlines results in an $O(n^2 \log n)$ algorithm, while Baseline is in $O(n^2)$ and EXACT$_u$ is linearathmic. This is what we observed across different settings, independent of the choice of the dataset. CFCQP did not finish for any of the settings for DoT dataset (Figure 9). For other datasets also, it had a significantly worse running time. For example, for weather dataset and $n = 42K$ tuples, CFCQP took 1709 seconds while BASELINE finished in 46 seconds and EXACT$_u$ run in 0.01 seconds. Having a quadratic runtime, Baseline is not scalable as it required more than half an hour to finish for the DoT dataset with 457K data records and 32 seconds for the 45K records in the Weather dataset. On the other hand, the linearithmic complexity of Algorithm 2 resulted an acceptable efficiency in all of these settings. The Exact algorithm finished in 7 milliseconds for CO2 dataset for 10K records, in 10 milliseconds for weather dataset, and in 2 seconds for the DoT dataset with 457K records. For the very large stock dataset, for 10 million records, it finished in 46 seconds.

7.3.2 Constrained Trendlines

Next we study the performance of Baseline (we adopted Algorithms 1 for constrained trendlines) and EXACT$_c$ (Algorithm 3). We selected the CO2 and Weather datasets for this experiment. Considering the overhead of the RBT implementation, we use $10 \log n$ as the $O(\log n)$ threshold in line 2 of Algorithm 3. Similar to the unconstrained trendlines experiments, first, we vary the input size (n), while setting the width of the window (the size of valid region for a point $b \in R(b)$) as 1000. The results are provides in Figures 11 and 12. In all settings Exact outperforms Baseline in an order of magnitude. Still, comparing the results with the ones for unconstrained trendlines, Algorithm 3 is slower than Algorithm 2. That is due to the overhead of RBT operations. Also Baseline has a better performance for constrained trendlines in comparison with unconstrained trendlines. That is because here there are less valid trendlines for Baseline to iterate over. As the next experiment, setting the input size as 20K, we vary the with of the valid window in $R(e)$ (Figures 13 and 14). While the performance of Baseline linearly depends on the width of the window, Algorithm 3 is logarithmic to the window size. This is confirmed in our results.

7.3.3 Randomized Algorithms

After studying the performance of the exact algorithms, next we move to randomized algorithms. Even though, our exact algorithms are linearithmic, they may require a few minutes for the very large settings. Using the stock dataset and PAIRSAMPLING as the randomized algorithm, while setting the sampling budget to $N = 10$K, we vary the input size from 2 million to 10 million and compared the performance of EXACT$_u$ and PAIRSAMPLING for computing support. The results are provided in Figures 15 and 16. While the exact algorithm required up to the order of a minute to compute the support, PAIRSAMPLING finished in at most one second. Still, comparing the dashed lines in Figure 16, its estimate of the support was almost the same as the exact values.

In § 5.2.2, we studied the variance of the POINTSAMPLING algorithm and showed that, even though it uses N^2 samples, its variance is not necessarily less than PAIRSAMPLING. Here, we also demonstrate an experimental comparison between these two algorithms. We used weather dataset for this experiment and considered the temperature in the New York city. In order to have an accurate comparison between the variances, for each setting we repeated each batch of 30 experiments, 30 times. For each batch we computed the variance and took the average of the 30 variances. We compare the variance of POINTSAMPLING against PAIRSAMPLING with (i) budget of N and (ii) budget of $N \log N$. While (i) represents the equal sampling budget comparison, (ii) represents equal computation cost (assuming $O(1)$ cost for sampling). Using the statements with various supports (discovered using Algorithm 6), we studied the impact of varying support (Figure 17). We also run an experiment for varying the sampling budget (Figure 18). In all experiments, for a fixed sampling budget, POINTSAMPLING

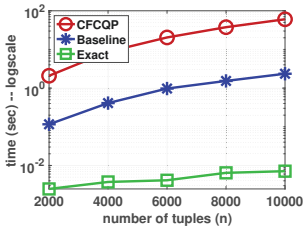

Figure 7: CO2 dataset, unconstrained trendlines, varying n

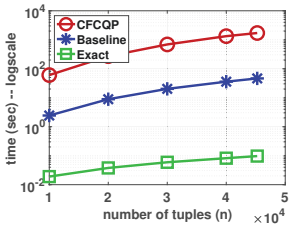

Figure 8: Weather dataset, unconstrained trendlines, varying n

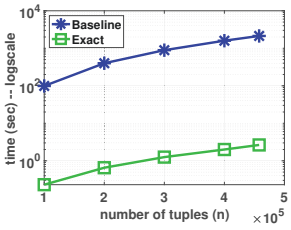

Figure 9: DoT dataset, unconstrained trendlines, varying n

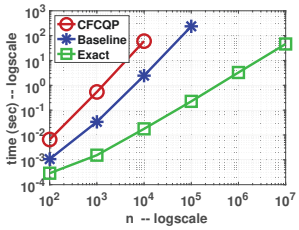

Figure 10: Stock dataset, unconstrained trendlines, varying n

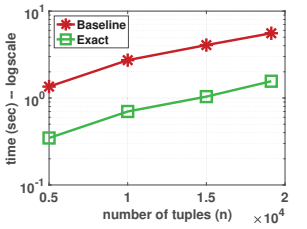

Figure 11: CO2 dataset, constrained trendlines, varying n

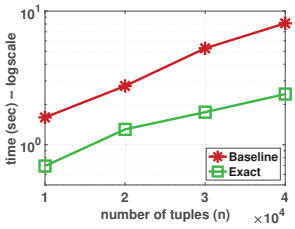

Figure 12: Weather dataset, constrained trendlines, varying n

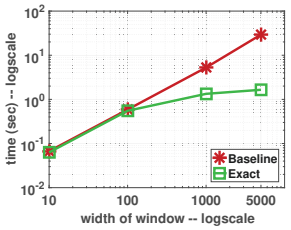

Figure 13: CO2 dataset, constrained trendlines, varying width of window

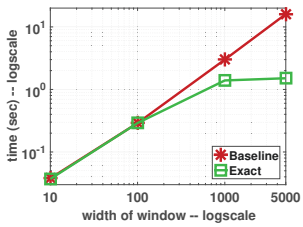

Figure 14: Weather dataset, constrained, varying width of window

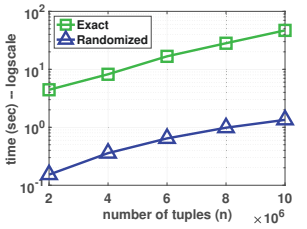

Figure 15: Stock dataset, scalability, Exact v.s. Randomized: efficiency

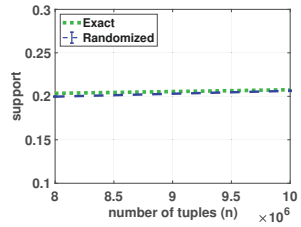

Figure 16: Stock dataset, scalability, Exact v.s. Randomized: effectiveness

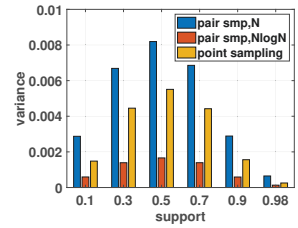

Figure 17: Weather dataset, New York city, Pair sampling v.s. Point sampling, varying support

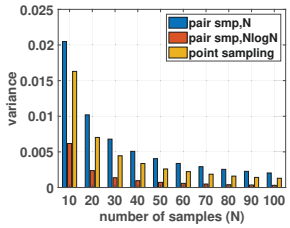

Figure 18: Weather dataset, New York city, Pair sampling v.s. Point sampling, varying budget

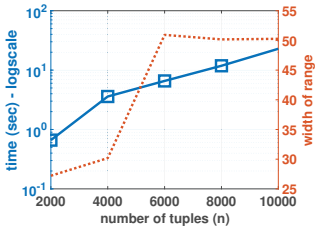

Figure 19: Weather dataset, New York city, tightest statement with support 0.8, varying n

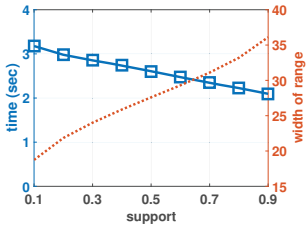

Figure 20: Weather dataset, New York city, tightest statement in Example 1, varying support

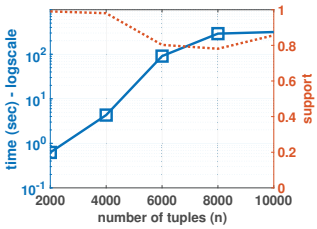

Figure 21: Weather dataset, New York city, most supported statement, varying n, $\alpha = 30$

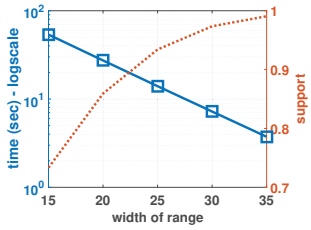

Figure 22: Weather dataset, New York city, most supported statement in Example 1, varying α

had a lower variance. This indicates that POINTSAMPLING is a good approach for the cases that sampling is costly. On the other hand, PAIRSAMPLING with the budget of $N \log N$ outperformed POINTSAMPLING in all cases. This means that when sampling is not costly (including the traditional dataset model) PAIRSAMPLING is preferred.

7.3.4 *Most Supported Statement*

Finally, we evaluate our proposed algorithms in § 6 for Problems 2 and 3. Again, as the default dataset, we used the weather dataset for these experiments, while using the temperature in the New York city as the target value. First, we use Algorithm 6 for finding the tightest statements for (i. Figure 19) varying n (while setting support to 0.8) and (ii. Figure 20) varying support. The left-y-axis in the plots (and the blue line with the square marker) show the time, while the right-y-axis (and the dashed line) show the width of the range for the tightest statement. The algorithm required a couple of seconds for finding the tightest statement with varying support values and up to 20 seconds for varying n. While the width of the tightest statement does not seem to depend on n, it clearly increases with the value of support. That is because we need to make the statements less restrictive such that more trendlines support it. Next, we use Algorithm 7 for finding the most supported statements for (i. Figure 21) varying n (while setting the statement range width to $\alpha = 30$) and (ii. Figure 22) varying width of the statement range. Clearly, as n increases the running time increases, as it took 315 seconds for the algorithm to finish for $n = 10$K. From the right-y-axis in Figure 21, the support value, as expected, does not depend on n, whereas (looking at Figure 22) as the width of the statement range increases the support value get close to 1.

8. RELATED WORK

Initially developed in journalism, computational fact checking aims to detect fake news, that is comparing the claims extracted from the news content against the existing facts [3, 13–19]. The initial fact checking efforts include manual methods based on the domain knowledge of human expert and crowdsourcing [16, 18]. Manual fact checking efforts, however, are not scalable and do not use the existing data. As a result, computational fact checking has emerged, where the ultimate goal is to have a platform that can automatically evaluate a claim in real-time [15]. Computational fact checking heavily rely on techniques form natural language processing [20, 21], information retrieval [22, 23], and graph theory [13]. A set of work in automated fact checking focus on knowledge extraction from different data sources [24–26], data cleaning and integration [27–29], and credibility evaluation [30, 31]. Existing work also includes style-based [32–35], propagation-based [36–38], and credibility-based [37, 39–42] study of fake news. Further information about fake news and the detection mechanisms can be found in a literature survey by Zhou and Zafarani [43].

Using perturbations for studying uncertainty has been studied in different context in data management [44–46]. Perturbation is an effective technique for studying the robustness of query outputs. For example, [47–49] use function perturbation for verifying the stability of ranking queries, as well as discovering fair and stable rankings. Query perturbation has also been used for retrieving more relevant query results [50–53]. The idea of query perturbation has also seen its applications in the context of the computational journalism, in both fact-checking [3] and lead-finding [54]. [54] has a different objective from our problem: finding a few representative points to capture the high-value regions of a complex surface. Besides, in this paper, we treat all points in the support region indifferently, and shape of the surface is not our focus. The focus of [3] has been on the modeling side– a generic framework for perturbation-based fact-checking. In contrast, this paper is on the technical side. Drilling down on the trendline statement, we address both checking and mining aspects. As observed in § 7.3, using [3] for computing support results in a worse performance than our baseline. The notion of "support", proposed in this paper, is a natural measure that can be defined within the framework and complementary to those defined in [3].

9. DISCUSSIONS

In this paper, we study statements made based on comparing a pair of points. Cherry-picking has a long history and hence many different forms. In a nice article at PolitiFact [55], L. Jacobson goes over some of the examples of cherry-picking in US politics. According to this article, PolitiFact has reported cherry-picking "hundreds of times" in their fact-checks. While we believe our notion of support can be adopted for all cherry-picking settings, how to efficiently compute the support is problem-specific. Of course, our problem formulation, while covering many, does not cover all forms of cherry-picking. Still, for many such settings, our algorithms can simply be adopted. In the following, looking into some real-life problems, we discuss if those fit our problem formulation and if not whether our proposal can be adjusted for those.

A large number of the cherry-picked statements are made by comparing a pair of points where our algorithms can directly (or after small preprocessing as explained in § 2.2) be applied. For example, consider President Trump's tweet, comparing his approval rate with president Obama [56]. He cherry-picked a single poll source and a specific date which shows the highest approval for him. To validate this statement, similar to § 7.2, one could compute the statement support for different times/poll-sources for President Trump v.s. President Obama. Another example is Trump's campaign ad. about undocumented immigrants. The ad. cherry-picked a single undocumented immigrant to picture the whole group "as dangerous", compared to native-born Americans. Of course, computing the support of the statement "undocumented immigrant are more dangerous (commit crime more) than native-born Americans" can numerically show how accurate it is. Yet other examples are the statement made by President Trump about income levels and unemployment numbers of African Americans being worst under president Obama than ever [57] or president Trump having the highest Poll Numbers in the history of the Republican Party [58], which can be evaluated by computing their supports.

Some statements are made based on a single point, rather than a pair of points. An example is a statement by the Democratic National Committee that no middle-class taxpayers stood to gain from President Trump's tax bill. Such cases can be viewed as special cases of constrained trendlines (by fixing $R(e)$ to a single point) where Algorithm 3 runs in $O(n)$. Another variation is when all trendlines are not equally important. For instance, the data in different time periods may have different contributions to the correctness of a statement. In such cases, instead of computing the simple average, one can consider a weighted average using a user-provided importance function (such as Gaussian decaying functions around the initial trendline points b and e). Let $w(\cdot)$ be the user-provided importance function. To adjust for these cases, Equation 5 shall be adapted to reflect the sum of weights, instead of counts, i.e., $F(y) = \sum_{\forall dx \in R(e) \mid y(dx) < y} w(dx)$. Equation 6 shall also be adapted accordingly to $w[i] = w(dx[i])\big(F\big(y(dx[i]) + \top\big) - F\big(y(dx[i]) + \bot\big)\big)$. Such modeling change would not affect the complexity of the proposed efficient algorithms.

We would like to reiterate that not all forms of cherry-picking are covered by our formulation. Sometimes, a trendline statement is not cherry-picked, but the narrative around it can be misleading. President Trump's tweet: "*Because of my policies*, Black Unemployment has just been reported to be at the lowest rate ever recorded" [59] is such an example. Similarly, he wrongly claimed credit for lack of commercial airline crash deaths during his presidency, while there have not been crash-deaths since 2013 [60]. Another case is cherry-picking sentences from natural language texts that cannot be directly evaluated with our method since our notion of support is based on numerical values. Still, in such cases, experts or NLP methods can be used for classifying sentences as if those support the overall statement and then compute their support.

Finally, we would like to emphasize that human-in-the-loop is necessary to transform "complicated" statements to the problem inputs and to identify and provide proper data to the system.

10. CONCLUSION

In this paper, we proposed a system for detecting cherry-picked trendlines. We defined a notion of support for this purpose and, formally defining terms, designed linearithmic exact algorithms for trendlines with different constraint models, followed by randomized approximation algorithms. We also studied the problem of finding mostly supported statements by data. Besides theoretical analysis, we conducted extensive experiments on real-world datasets that confirm the validity, efficiency, and effectiveness of our proposal In this paper we proposed sampling-based approximation for scalability. We will consider designing massively parallel algorithms, similar to [61], as part of our future work.

11. REFERENCES

[1] Claire Wardle. Fake news. its complicated. *First Draft News*, 16, 2017.

[2] John Mason. How to use short timeframes to distort reality: a guide to cherrypicking. www.skepticalscience.com/cherrypicking-guide.html, 25 September 2013.

[3] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Computational fact checking through query perturbations. *ACM Transactions on Database Systems (TODS)*, 42(1):4, 2017.

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[5] Christian P Robert. *Monte carlo methods*. Wiley Online Library, 2004.

[6] Fred J Hickernell, Lan Jiang, Yuewei Liu, and Art B Owen. Guaranteed conservative fixed width confidence intervals via monte carlo sampling. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 105–128. Springer, 2013.

[7] Rick Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

[8] Carbon dioxide levels in the atmosphere (ppm on a daily basis). github.com/datasets/co2-ppm-daily.

[9] Earth system research laboratory. www.esrl.noaa.gov/.

[10] Historical hourly weather data 2012-2017, kaggle. www.kaggle.com/selfishgene/historical-hourly-weather-data/home. Accessed: 2018.

[11] Us department of transportation. www.transtats.bts.gov/DL$_{S}elect Fields.asp$? Accessed: 2018.

[12] Evan Hallmark. Daily historical stock prices (1970 - 2018). www.kaggle.com/ehallmar/daily-historical-stock-prices-1970-2018/.

[13] Sarah Cohen, James T Hamilton, and Fred Turner. Computational journalism. *Communications of the ACM*, 54(10):66–71, 2011.

[14] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Toward computational fact-checking. *PVLDB*, 7(7):589–600, 2014.

[15] Naeemul Hassan, Bill Adair, James T Hamilton, Chengkai Li, Mark Tremayne, Jun Yang, and Cong Yu. The quest to automate fact-checking. In *Proceedings of the 2015 Computation+Journalism Symposium*, 2015.

[16] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1803–1812. ACM, 2017.

[17] Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph, Aaditya Kulkarni, Anil Kumar Nayak, et al. Claimbuster: the first-ever end-to-end fact-checking system. *PVLDB*, 10(12):1945–1948, 2017.

[18] Naeemul Hassan, Chengkai Li, and Mark Tremayne. Detecting check-worthy factual claims in presidential debates. In *Proceedings of the 24th acm international on conference on information and knowledge management*, pages 1835–1838. ACM, 2015.

[19] Naeemul Hassan, Afroza Sultana, You Wu, Gensheng Zhang, Chengkai Li, Jun Yang, and Cong Yu. Data in, fact out: automated monitoring of facts by factwatcher. *PVLDB*, 7(13):1557–1560, 2014.

[20] Yunyao Li, Ishan Chaudhuri, Huahai Yang, Satinder Singh, and HV Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1165–1168. ACM, 2007.

[21] Yunyao Li, Huahai Yang, and HV Jagadish. Constructing a generic natural language interface for an xml database. In *International Conference on Extending Database Technology*, pages 737–754. Springer, 2006.

[22] Philip A Bernstein and Laura M Haas. Information integration in the enterprise. *Communications of the ACM*, 51(9):72–79, 2008.

[23] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[24] Sachin Pawar, Girish K Palshikar, and Pushpak Bhattacharyya. Relation extraction: A survey. *arXiv preprint arXiv:1712.05191*, 2017.

[25] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[26] Ralph Grishman. Information extraction. *IEEE Intelligent Systems*, 30(5):8–15, 2015.

[27] Rebecca C Steorts, Rob Hall, and Stephen E Fienberg. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association*, 111(516):1660–1672, 2016.

[28] Amr Magdy and Nayer Wanas. Web-based statistical fact checking of textual documents. In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, pages 103–110. ACM, 2010.

[29] Yasser Altowim, Dmitri V Kalashnikov, and Sharad Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[30] Diego Esteves, Aniketh Janardhan Reddy, Piyush Chawla, and Jens Lehmann. Belittling the source: Trustworthiness indicators to obfuscate fake news on the web. *arXiv preprint arXiv:1809.00494*, 2018.

[31] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *PVLDB*, 8(9):938–949, 2015.

[32] Gary D Bond, Rebecka D Holman, Jamie-Ann L Eggert, Lassiter F Speller, Olivia N Garcia, Sasha C Mejia, Kohlby W Mcinnes, Eleny C Ceniceros, and Rebecca Rustige. lyin'ted,crooked hillary, and deceptive donald: Language of lies in the 2016 us presidential debates. *Applied Cognitive Psychology*, 31(6):668–677, 2017.

[33] Svitlana Volkova, Kyle Shaffer, Jin Yea Jang, and Nathan Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 647–653, 2017.

[34] Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek

Bevendorff, and Benno Stein. A stylometric inquiry into hyperpartisan and fake news. *arXiv preprint arXiv:1702.05638*, 2017.

[35] Dina Pisarevskaya. Deception detection in news reports in the russian language: Lexics and discourse. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 74–79, 2017.

[36] Jing Ma, Wei Gao, and Kam-Fai Wong. Rumor detection on twitter with tree-structured recursive neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1980–1989, 2018.

[37] Ke Wu, Song Yang, and Kenny Q Zhu. False rumors detection on sina weibo by propagation structures. In *2015 IEEE 31st international conference on data engineering*, pages 651–662. IEEE, 2015.

[38] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[39] Zhiwei Jin, Juan Cao, Yongdong Zhang, and Jiebo Luo. News verification by exploiting conflicting social viewpoints in microblogs. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[40] Manish Gupta, Peixiang Zhao, and Jiawei Han. Evaluating event credibility on twitter. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 153–164. SIAM, 2012.

[41] Jiawei Zhang, Limeng Cui, Yanjie Fu, and Fisher B Gouza. Fake news detection with deep diffusive network model. *arXiv preprint arXiv:1805.08751*, 2018.

[42] Kai Shu, Suhang Wang, and Huan Liu. Exploiting tri-relationship for fake news detection. *arXiv preprint arXiv:1712.07709*, 2017.

[43] Xinyi Zhou and Reza Zafarani. Fake news: A survey of research, detection methods, and opportunities. *arXiv preprint arXiv:1812.00315*, 2018.

[44] Charu C Aggarwal. *Managing and mining uncertain data*, volume 35. Springer Science & Business Media, 2010.

[45] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter J Haas. The monte carlo database system: Stochastic analysis close to the data. *ACM Transactions on Database Systems (TODS)*, 36(3):18, 2011.

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

[47] Abolfazl Asudeh, HV Jagadish, Gerome Miklau, and Julia Stoyanovich. On obtaining stable rankings. *PVLDB*, 12(3):237–250, 2018.

[48] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1259–1276. ACM, 2019.

[49] Yifan Guan, Abolfazl Asudeh, Pranav Mayuram, HV Jagadish, Julia Stoyanovich, Gerome Miklau, and Gautam Das. Mithraranking: A system for responsible ranking design. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1913–1916. ACM, 2019.

[50] Surajit Chaudhuri. Generalization and a framework for query modification. In *[1990] Proceedings. Sixth International Conference on Data Engineering*, pages 138–145. IEEE, 1990.

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

[52] S-Y Huh, K-H Moon, and Heeseok Lee. A data abstraction approach for query relaxation. *Information and Software Technology*, 42(6):407–418, 2000.

[53] Christian S Jensen and Richard Snodgrass. Temporal specialization and generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974, 1994.

[54] You Wu, Junyang Gao, Pankaj K Agarwal, and Jun Yang. Finding diverse, high-value representatives on a surface of answers. *PVLDB*, 10(7):793–804, 2017.

[55] Louis Jacobson. The age of cherry-picking. web.archive.org/web/20180715230354/http://www.politifact.com/truth-o-meter/article/2018/feb/05/age-cherry-picking/, 5 Feb. 2018.

[56] Louis Jacobson. Donald trump tweet on 50% approval cherry-picks polling data. web.archive.org/web/20190626042240/https://www.politifact.com/truth-o-meter/statements/2017/jun/19/donald-trump/donald-trump-tweet-50-approval-cherry-picks-pollin/, 19 June 2017.

[57] Louis Jacobson. Donald trump: Black income, unemployment 'worse now than just about ever'. web.archive.org/web/20191101151207/https://www.politifact.com/truth-o-meter/statements/2015/aug/02/donald-trump/donald-trump-black-income-unemployment-worse-now-j/, 2 Aug. 2015.

[58] Louis Jacobson. No, donald trump's poll numbers do not beat lincoln, all other GOP presidents. web.archive.org/web/20191204110810/https://www.politifact.com/truth-o-meter/statements/2018/jul/30/donald-trump/has-donald-trump-had-highest-poll-numbers-any-gop-, 30 July 2018.

[59] Louis Jacobson. Donald trump is partly correct in response to jay-z about black unemployment. web.archive.org/web/20191030053546/https://www.politifact.com/truth-o-meter/statements/2018/jan/30/donald-trump/donald-trump-partly-correct-rejoinder-jay-z/, 30 Jan. 2018.

[60] Joan Lowy. Fact check: Trump wrongly claims credit for lack of commercial airline crash deaths. web.archive.org/web/20190918010302/https://www.chicagotribune.com/nation-world/ct-trump-airline-safety-fact-check-20180102-story.html, 2 Jan. 2018.

[61] Yufei Tao. Massively parallel entity matching with linear classification in low dimensional space. In *21st International Conference on Database Theory (ICDT 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.