Apr 7, 2025 Lecture Note: Bias in Cherry-picked Data Presentation

[Devendra Seelamneni, Aryan Rao Neelagiri]

CS 516: Responsible Data Science and Algorithmic Fairness; Spring 2025
Abolfazl Asudeh; www.cs.uic.edu/∼ asudeh/teaching/archive/cs516spring25/

1 Introduction

Cherry-picking involves selecting specific data points or timeframes along with weighting schemes to support a preferred outcome rather than reveal genuine realities. Such practices produce political narrative distortions by focusing on single trends while adjusting weights in university and product rankings and orchestrating news presentation sequences to subtly influence public perspectives. The analysis evaluates news ordering together with trendline statements and linear rankings through a perturbation-based support measure which determines the stability of conclusions against alternative scenarios. Both systematic approaches determine the stability of original results through perturbation analysis while Monte Carlo sampling provides efficient confidence-bound estimates for impossible exhaustive checks. The optimization procedures find the most typical trends and rankings together with story sequences that satisfy average-case neutrality and worst-case bias constraints. The tools establish a systematic framework to identify and fight against cherry-picking in multiple applications.

2 Common Preliminaries & Support Framework

1. **Region of Interest (U).** The universe of all valid alternatives for a given analysis:
   - *Trendlines:* all (begin, end) date pairs within a president's term.
   - *Rankings:* all weight vectors within prescribed bounds.
   - *News feeds:* all permutations of the headline list.
2. **Perturbation-Based Support.** Given reported outcome O and region U, define
$$\omega_U(O) = \frac{\left|\{\, u \in U \mid O(u) \text{ remains equivalent to } O\}\right|}{|U|}.$$
   A low $\omega_U(O)$ signals cherry-picking.
3. **Statistical Estimation via Sampling.** When $|U|$ is too large to enumerate, draw N random elements $u_i \in U$. Let
$$X_i = \begin{cases} 1, & O(u_i) \text{ supports } O, \\ 0, & \text{otherwise.} \end{cases} \qquad \hat{\omega} = \frac{1}{N}\sum_{i=1}^{N} X_i, \quad e = Z_{1-\frac{\alpha}{2}}\sqrt{\frac{\hat{\omega}(1-\hat{\omega})}{N}}.$$
   Then $[\hat{\omega} - e,\ \hat{\omega} + e]$ is a $(1-\alpha)\%$ confidence interval for $\omega_U(O)$.

3 Cherry-Picked News Ordering

When two stories appear back-to-back, the first can *prime* opinions about the second—an effect known as *opinion priming*. A *cherry-picked ordering* exploits this by placing particular headlines adjacently to induce a desired overall impression. The twin objectives are:

1. **Detection:** Decide if a given ordering exhibits unusually strong priming compared to random shuffles.
2. **Resolution:** Find a reordering that minimizes priming under average-case or worst-case criteria.

3.1 Model & Formal Definitions

- **POP function** $\phi\colon t \times t \to [0,1]$**.** $\phi(C_i, C_j)$ measures intrinsic priming when C_i, C_j are adjacent.
- **Decay** $\delta\colon \mathbb{N} \to [0,1]$**.** $\delta(d)$ models attenuation with distance d, $\delta(1) = 1$.
- **Pairwise neutrality**

$$\eta_{ij} = 1 - \delta\big(|s(C_i) - s(C_j)|\big)\,\phi(C_i, C_j).$$

For $t = \{C_1, \ldots, C_n\}$ and ordering s, the neutrality under aggregation agg is

$$\mathrm{Neut}_{\mathrm{agg}}(s) = \mathrm{agg}\{\eta_{ij} \mid 1 \le i < j \le n\}.$$

3.2 Cherry-Picking Detection

1. Sample A random permutations via Fisher–Yates.
2. Compute $\nu_k = \mathrm{Neut}_{\mathrm{agg}}(s_k)$, obtain sample mean $\bar{\nu}$ and standard deviation f.
3. Let $\nu^* = \mathrm{Neut}_{\mathrm{agg}}(s)$. By the Saw–Yang–Mo inequality,

$$\lambda = \frac{|\nu^* - \bar{\nu}|}{f}\sqrt{\frac{A}{A+1}}, \quad \Pr\big(|\nu^* - \bar{\nu}| \ge f\,\tfrac{\lambda}{\sqrt{A/(A+1)}}\big) \le \frac{1}{\lambda^2} + \frac{1}{A}.$$

4. Conclude s is cherry-picked if the right-hand side is below your tolerance.

3.2.1 Example: Detection on Six Items

- Consider the presented ordering

$$s_0 = \langle t_1,\, t_3,\, t_4,\, t_2,\, t_5,\, t_6 \rangle.$$

  Its average-neutrality score is $\mathrm{Neut}_{\mathrm{avg}}(s_0) = 0.62$.

- Draw $A = 20$ random shuffles. Compute their neutralities: $\bar{\nu} \approx 0.82,\ f \approx 0.05$.
- Compute $\lambda \approx \frac{|0.62-0.82|}{0.05}\sqrt{\frac{20}{21}} \approx 8.7$ so the upper bound on the tail probability is $\le 1/8.7^2 + 1/20 \approx 0.03$.
- Since 3% is below a typical significance threshold, conclude s_0 is likely cherry-picked.

3.3 Maximizing Neutrality Under Average Aggregation

The *average* aggregation scores an ordering by computing the pairwise priming risk for *every* pair of headlines and then averaging these values. A perfectly neutral feed therefore attains a score of 1, indicating that no two stories prime each other. Maximising this measure is equivalent to finding a Hamiltonian path whose total edge-weight (sum of pairwise neutralities) is PathMaxTSP, which is NP-hard. To tackle this, we were introduced to three approximation algorithms ApproxMat, ApproxCC, and Approx3CC; these efficiently construct near-optimal orderings.

3.3.1 PathMaxTSP

Finding a Hamiltonian path of maximum total neutrality (sum of edge-weights) is NP-hard. We have three approximation algorithms.

Iterated Matching (ApproxMat) ApproxMat executes a max-weight matching on the current graph, then merges each matched pair into a "super-node" path. Any unmatched vertex (when $|V|$ is odd) is carried forward unchanged. It then rebuilds a smaller graph whose nodes are these super-nodes, with edge-weights equal to the maximum neutrality between their endpoints, and repeats until a single super-node remains. That final super-node implicitly encodes a Hamiltonian path. The very first matching already captures at least 50% of the optimum, and the full procedure runs in $O(n^4)$ time using Edmonds' blossom algorithm for matching.

Example 3.3.1.1

1. **Iteration 1:** Graph on $\{t_1, \ldots, t_6\}$ has pairwise neutrality weights as in Figure 1a. Compute max-weight matching

$$\{(t_1, t_3) = 0.8,\ (t_2, t_4) = 0.3,\ (t_5, t_6) = 1.0\}.$$

   Merge each pair into super-nodes $S_A = [t_1, t_3],\ S_B = [t_2, t_4],\ S_C = [t_5, t_6]$.

2. **Iteration 2:** Build graph on $\{S_A, S_B, S_C\}$, with edge-weights the maximum neutrality between endpoints. The max-weight matching is (S_B, S_C) via edge $(t_4, t_5) = 1.0$. Merge into

$$S_D = [\, t_2, t_4, t_5, t_6 \,],$$

   leaving $\{S_A, S_D\}$.

3. **Iteration 3:** Only matching is (S_A, S_D) via $(t_1, t_2) = 1.0$. Orient and merge to obtain the final path

$$\langle t_3,\ t_1,\ t_2,\ t_4,\ t_5,\ t_6 \rangle.$$

Its adjacencies have weights $\{0.7, 1.0, 0.3, 1.0, 1.0\}$, summing to 4.1, so

$$\text{Neut}_{\text{avg}} = 4.1/5 = 0.82.$$

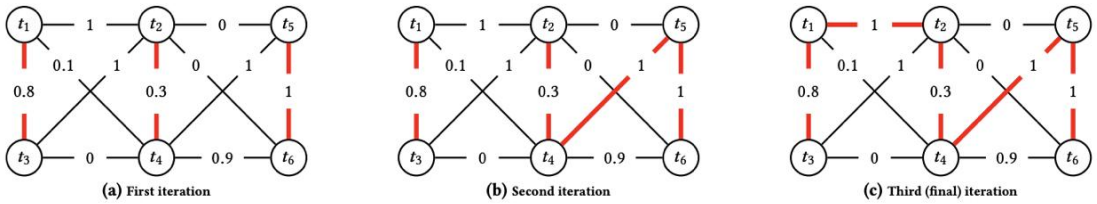

Figure 1: ApproxMat iterations. (a) initial matching, (b) second iteration, (c) third iteration.

Iterated Cycle-Cover (ApproxCC) ApproxCC begins by computing a max-weight cycle cover in $O(n^3)$ time via reduction to a bipartite matching (Hungarian method). It then deletes the lightest edge from each cycle producing disjoint paths and treats each path as a super-node, and reconnects them by their heaviest endpoint edges to form a new condensed graph. Repeating until one super-node remains yields a Hamiltonian path. By removing only the weakest edge in each cycle, it retains at least 50% of the maximum weight and runs in $O(n^3)$, improving on ApproxMat's $O(n^4)$.

Example 3.3.1.2

1. **Iteration 1:** Max-weight cycle cover $\{(t_1, t_2, t_3), (t_4, t_5, t_6)\}$. Remove the lightest edge from each cycle (e.g. (t_1, t_2) and (t_5, t_6)) to form paths.

2. **Iteration 2:** On the four resulting super-nodes, compute the max-weight cycle cover and remove its lightest link (e.g. (t_6, t_3)), yielding the single super-node whose internal order is $\langle t_3,\ t_1,\ t_2,\ t_4,\ t_5,\ t_6 \rangle$.

Its adjacencies have weights $\{0.7, 1.0, 0.3, 1.0, 1.0\}$, summing to 4.1, so

$$\text{Neut}_{\text{avg}} = \frac{4.1}{5} = 0.82.$$

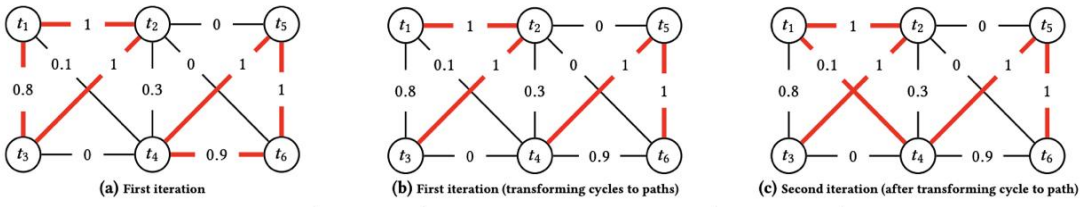

Figure 2: ApproxCC on six items.

3-Cycle-Cover (Approx3CC) Approx3CC is analogous to ApproxCC but restricts its cycle cover to cycles of length at least 3. One reduces the 3-cycle-cover problem to a larger matching instance, solves it in $O(n^7)$ time, removes the lightest edge from each 3-cycle to form paths, and merges as before. This extra restriction raises the approximation guarantee from $\frac{1}{2}$ to $2/3$, at the cost of higher $O(n^7)$ runtime.

Algorithm:

1. *3-cycle-cover reduction:* Build an expanded auxiliary graph (via Tutte/Eppstein gadgetry) in which each perfect matching corresponds one-to-one with a 3-cycle cover in the original graph.
2. *Solve matching:* Run Edmonds' blossom algorithm on the auxiliary graph in $O(n^7)$ to obtain a max-weight matching, hence a max-weight 3-cycle cover.
3. *Edge pruning:* In each 3-cycle, delete its lightest edge and turning every cycle into a path of length 2.
4. *Path concatenation:* Arbitrarily join the resulting small paths end-to-end to form a single Hamiltonian path. This path has total weight at least $\frac{2}{3} \cdot \mathrm{OPT}$.

Trade-off: The improved 2/3 approximation guarantee incurs a steep $O(n^7)$ time cost. By contrast, ApproxMat runs in $O(n^4)$ and ApproxCC in $O(n^3)$, so Approx3CC is primarily of theoretical interest or only viable on small graphs.

3.4 Maximizing Neutrality Under Min Aggregation: PathMaxScatterTSP

The minimum-edge neutrality objective is finding a Hamiltonian path whose weakest adjacency is as large as possible, defines the PathMaxScatterTSP. As no polynomial-time constant-factor approximation exists (unless P=NP), we were proposed by a practical heuristic that binary-searches an edge-neutrality threshold X and uses a fast local-search oracle to test feasibility.

Algorithm

1. **Threshold test (`IsFeasible`):** Given X, form graph G' on the same vertices where each edge of original neutrality w has cost

$$\mathrm{cost}(e) = \begin{cases} 0, & w \geq X, \\ X - w, & w < X. \end{cases}$$

   A zero-cost Hamiltonian cycle in G' implies every edge in the corresponding cycle in G has neutrality $\geq X$.

2. **Local search (2-opt):** Run the classic 2-opt TSP heuristic on G' to seek a zero-cost cycle. Because 2-opt is "any-time," it can be halted early if desired.
3. **Binary search over X:** Sort the distinct original neutralities $\{w_e\}$. Use binary search on this sorted list, calling `IsFeasible` at each midpoint, to find the largest X for which a zero-cost cycle is found.
4. **Path extraction:** Given the zero-cost cycle at threshold X, remove its lightest edge to obtain a Hamiltonian path whose minimum adjacency neutrality is $\geq X$.
5. **Early-cut option:** Optionally cap 2-opt at k swaps for faster but potentially slightly sub-optimal X.

Empirical performance This heuristic is the first effective solver for PathMaxScatterTSP in practice. By trading off local-search effort against threshold precision, it outperforms random sampling, achieves near-optimal weakest-edge values on graphs up to $n = 70$, and allows users to dial run-time versus solution quality via the 2-opt iteration limit.

4 Cherry-Picked Trendlines

Trendlines are fundamental tools to summarize and visualize the change of a variable over time. In many fields like economics, epidemiology, and political analysts draw a line between two dates to "tell the story." However, when an analyst deliberately selects those two dates to make a trend appear stronger (or weaker) than the overall data supports, this is known as *cherry-picking*.

4.1 Intuition and Pitfalls

A single trendline can be very misleading if the data have high variability. For example, unemployment rates might fluctuate seasonally; choosing a high initial point in winter and a low ending point in summer exaggerates a "drop." Without examining other pairs, the audience is misled.

4.2 Formal Definitions

Let $D = \{(t_i, y_i)\}$ be a time series, where t_i are sorted time points and y_i are observations.
A *trendline* $\theta = (b, e)$ uses two indices $b < e$, connecting (t_b, y_b) to (t_e, y_e).
A *statement* S_θ asserts a change:

$$y_e - y_b \in (\bot, \top),$$

where $(\bot, \top)$ defines a condition (e.g., negative for a "drop").

4.3 Support Measure

To quantify how representative a chosen trendline is, define the *support region*

$$R_S = R(b) \times R(e),$$

where $R(b)$, $R(e)$ are sets of valid start/end indices (e.g., all dates within a presidential term). The *support* of statement S is

$$\omega_{R_S}(S, D) = \frac{\left|\{(i, j) \in R(b) \times R(e) : y_j - y_i \in (\bot, \top)\}\right|}{|R(b)| \cdot |R(e)|}.$$

A small ω (e.g. < 0.1) indicates that under most valid alternatives, the statement fails—an alarm for cherry-picking.

4.4 Detection Algorithm ($O(n \log n)$)

Rather than enumerating all $|R(b)| \times |R(e)|$ pairs, we sort and binary-search:

Step 1: Sort values $\{y_j : j \in R(e)\}$ into an array $F[1..m]$ in ascending order.

Step 2: For each $i \in R(b)$:

- Compute bounds $L = y_i + \bot$ and $H = y_i + \top$.
- Find via binary search the first index ℓ with $F[\ell] \geq L$, and the last index r with $F[r] \leq H$.
- The count of supporting ends is $w_i = \max(0, r - \ell + 1)$.

Step 3: Sum $W = \sum_{i \in R(b)} w_i$. The support is $\omega = W/(|R(b)|\,|R(e)|)$.

4.5 Conceptual View: Support Computation Using Trendline Regions

To build a deeper intuition about trendline-based support, consider a time series where we wish to analyze how many potential trendlines satisfy a desired change condition. Specifically, we examine how regions of valid start and end points can be used to count the proportion of trendlines that meet a threshold for instance, a minimum increase of α units over time.

Let $y(p)$ denote the value at point p. For each $p \in R(b)$, we want to count the number of $p' \in R(e)$ such that:

$$y(p') - y(p) \geq \alpha.$$

We define this set as $R_p(e)$, the supporting end points for a specific start p. The baseline algorithm computes the total number of such valid pairs by iterating over all start points and checking against all end points, resulting in $O(n^2)$ time.

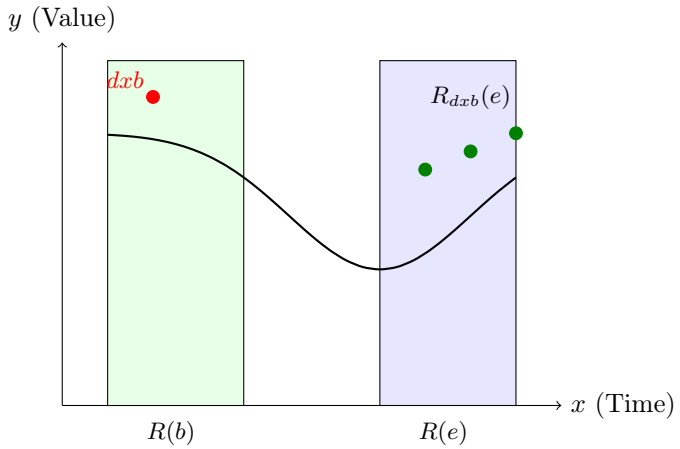

Figure 3: For a given point $dxb \in R(b)$, all points $dxe \in R(e)$ where $y(dxe) - y(dxb) \geq \alpha$ form the support region $R_{dxb}(e)$.

To optimize this, we exploit the structure: if $y(dxb_1) < y(dxb_2)$, then $R_{dxb_1}(e) \supseteq R_{dxb_2}(e)$. That is, lower-valued start points are likely to have more supporting ends.

Instead of re-evaluating from scratch for each start point, we can process them in increasing y order and incrementally build the support counts. Even better, we use a *cumulative function* F:

$$F(y) = |\{p' \in R(e) \mid y(p') < y\}|.$$

Then for a given $p \in R(b)$, the number of supporting ends is:

$$w_p = |R(e)| - F(y(p) + \alpha).$$

This allows efficient binary search over sorted y values from $R(e)$, giving overall runtime $O(n \log n)$. The total support is then:

$$\omega = \frac{1}{|R(b)||R(e)|} \sum_{p \in R(b)} w_p.$$

This optimization forms the core of the detection algorithm described earlier.

4.6 Illustrative Example

We consider an example to demonstrate how to compute the support of a statement about a drop in unemployment rate:

- Start points ($R(b)$): **Jan, Feb, Mar** with rates $7.2, 7.0, 6.9$
- End points ($R(e)$): **Jul, Aug, Sep** with rates $6.3, 6.4, 6.6$
- Target condition: $y(e) - y(b) \leq -0.5$ (i.e., drop of at least 0.5)

For each start point, we compare against each end point to check if the drop meets the condition:

- **Jan (7.2)**: All of Jul (6.3), Aug (6.4), Sep (6.6) satisfy $y(e) \leq 6.7 \rightarrow$ **3 matches**
- **Feb (7.0)**: Jul (6.3), Aug (6.4) satisfy $y(e) \leq 6.5 \rightarrow$ **2 matches**
- **Mar (6.9)**: Jul (6.3), Aug (6.4) satisfy $y(e) \leq 6.4 \rightarrow$ **2 matches**

Total matches $= 3 + 2 + 2 = 7$. Total combinations $= 3 \times 3 = 9$. Thus,

$$\omega = \frac{7}{9} \approx 0.78$$

This support score is fairly high, meaning the trendline from Jan to Jul is not cherry-picked; many other date combinations also support the same "drop" statement.

4.7 Resolution via Sliding Window ($O(\mathbf{n}^2)$)

To find the *most* supported statement, enumerate all differences

$$L = \{y_j - y_i : (i, j) \in R(b) \times R(e)\},$$

sort L, then slide a width-d window (where $d = \top - \bot$) to maximize the number of elements inside. This identifies the change interval most representative of the entire region.

5 Cherry-Picked Rankings

Rankings are commonly derived from weighted combinations of features, be it in university comparisons, product scores, or performance assessments. However, the final ranking is highly sensitive to the weight vector used. This raises a critical question: could a particular ranking be the result of cherry-picked weights rather than a fair evaluation? To address this, we analyze the *angular stability* of rankings by studying how the order changes as the weight direction varies in the dual space.

5.1 Motivation and Dual-Space Geometry

Given $d = 2$ features, each item i with values (x_{i1}, x_{i2}) maps to a line

$$x_{i1}w_1 + x_{i2}w_2 = 1$$

in the weight space (w_1, w_2). A ranking corresponds to ordering these lines by distance along a ray from the origin. As the ray angle θ rotates, the ranking changes only at *flip angles* where two lines intersect.

5.2 Flip Angle Computation

For adjacent items i and j, solve:

$$w_1x_{i1} + w_2x_{i2} = w_1x_{j1} + w_2x_{j2} \implies \theta_{ij} = \arctan \frac{x_{j1} - x_{i1}}{x_{i2} - x_{j2}}.$$

Each θ_{ij} marks a boundary of the region where the ranking order of i and j flips.

5.3 Support Measure

Let $U = [\theta_b, \theta_e]$ be the allowed angle range. The support of a fixed ranking r is

$$\omega_U(r) = \frac{\theta_e(r) - \theta_b(r)}{\theta_e - \theta_b},$$

where $[\theta_b(r), \theta_e(r)]$ is the maximal contiguous angle interval over which r remains constant.

5.4 Detection Algorithm (O(n))

Compute all needed $\theta_{i,i+1}$ for adjacent pairs in an observed ranking r:

1. For ranking $r = (t_1, ..., t_n)$, compute $\theta_{i,i+1}$ if t_i and t_{i+1} are incomparable.
2. Set $\theta_{\min}$ = max of all lower-bound angles, and $\theta_{\max}$ = min of upper-bound angles.
3. Support is $(\theta_{\max} - \theta_{\min})/(\theta_e - \theta_b)$.

5.5 Illustrative Example: Cherry-Picked Ranking

Consider a company that wants to rank five real estate agents based on two attributes:

- x_1: Customer Satisfaction
- x_2: Sales Volume

The company uses a linear scoring function:

$$f(t) = w_1 \cdot x_1 + w_2 \cdot x_2$$

with weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$, giving slightly more importance to sales.

The computed scores and resulting ranking are as follows:

Agent	**Customer Satisfaction** (x_1)	**Sales** (x_2)	**Score** $f(t)$
t1	0.63	0.71	1.34
t2	0.83	0.65	1.48
t3	0.58	0.78	1.36
t4	0.70	0.68	1.38
t5	0.53	0.82	1.35

Table 1: Agent attributes and scores using $f(t) = 1.1x_1 + 1.3x_2$

The induced ranking is:

$$t2 > t4 > t3 > t5 > t1$$

While this ranking is valid, its stability is questionable. Slightly changing the weights (e.g., using $\vec{w'} = \langle 1.2, 1.2 \rangle$) can shift the entire order. This sensitivity suggests the original ranking might be *cherry-picked*.

Geometric Interpretation (Dual Space)

In the dual space:

- Each agent t becomes a line: $d(t) : t[1]x_1 + t[2]x_2 = 1$
- The scoring function becomes a ray from the origin in the direction $\vec{w}$
- The order in which the ray intersects the lines determines the ranking

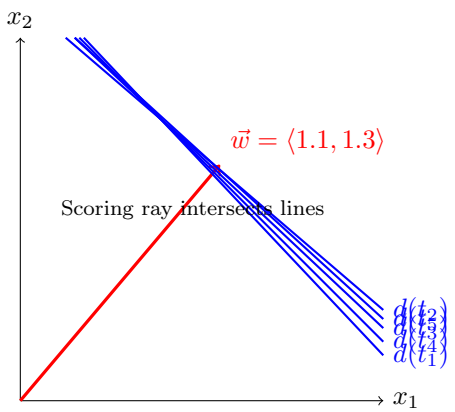

Figure 4: Dual space: agents as lines; rankings determined by order of intersections

Support of a Ranking

The ranking generated by $\vec{w}$ lies within a narrow region in weight space. We define:

$$\omega_U(r) = \frac{\text{Volume of Region Producing } r}{\text{Total Region of Interest } U}$$

A low support value implies the ranking only occurs under very specific weight choices indicating potential cherry-picking. In this case, the ranking has low support, as even a small change in $\vec{w}$ alters the outcome.

This geometric analysis makes it possible to detect unstable (and thus cherry-picked) rankings without needing to enumerate all weights.

5.6 Resolution Algorithm ($O(\mathbf{n}^2 \log n)$)

To find the *most* supported ranking:

1. Compute all flip angles θ_{ij} among item pairs.
2. Insert them into a min-heap sorted ascending.
3. Initialize ranking at θ_b, and sweep by popping events:
   - At each θ, swap the involved items in the current ranking.
   - Track the angular span until the next flip.
4. Return the ranking with maximal total span.

6 Sampling-Based Approximation

Exact methods can be computationally costly when $|R|$ or d is large. Monte Carlo sampling offers efficient approximate support estimates.

6.1 Trendline Sampling

Sample N random pairs (i, j) uniformly from $R(b) \times R(e)$. Define indicator:

$$X_k = \begin{cases} 1 & y_j - y_i \in (\perp, \top), \\ 0 & \text{otherwise.} \end{cases}$$

Then estimate

$$\hat{\omega} = \frac{1}{N} \sum_{k=1}^{N} X_k, \quad e = Z_{1-\alpha/2} \sqrt{\tfrac{\hat{\omega}(1-\hat{\omega})}{N}}.$$

6.2 Ranking Sampling

For $d > 2$ features, sample N weight vectors w_k from a spherical cone in $\mathbb{R}^d$ around a nominal direction. For each, compute ranking r_k and tally frequencies. The top ranking r^* has empirical support

$$\hat{\omega}(r^) = \frac{\#\{k : r_k = r^\}}{N}, \quad e = Z_{1-\alpha/2} \sqrt{\tfrac{\hat{\omega}(1-\hat{\omega})}{N}}.$$

7 Conclusion

A unified perturbation-based support metric quantifies cherry-picking across news ordering, trendline statements, and linear rankings. Neutrality maximization is cast as two graph problems PathMaxTSP (average-case) and PathMaxScatterTSP (worst-case) with $\frac{1}{2}$- and $\frac{2}{3}$-approximation or heuristic solvers. Trendline support is computed in $O(n \log n)$ via binary search (or $O(n^2)$ by a sliding-window), and ranking stability via angular-span analysis. Monte Carlo sampling delivers fast, confidence-bounded estimates at scale. Extending to richer decay models, adaptive sampling for extreme supports, and high-dimensional or multi-objective orderings will broaden applicability and strengthen defenses against subtle data manipulations.