Querying with Conflicts of Interest

Nischal Aryal
Oregon State University
Corvallis, USA
aryaln@oregonstate.edu

Arash Termehchy
Oregon State University
Corvallis, USA
termehca@oregonstate.edu

Marianne Winslett
University of Illinois
Urbana-Champaign, USA
winslett@illinois.edu

ABSTRACT

Conflicts of interest often arise between data sources and their users regarding how the users' information needs should be interpreted by the data source. For example, an online product search might be biased towards presenting certain products higher in its list of results to improve its revenue, which may not follow the user's desired ranking expressed in their query. The research community has proposed schemes for data sources to implement to ensure unbiased results. However, data sources usually have little or no incentive to implement these measures, e.g., their biases often increase their profits. Thus, we propose a novel framework for querying in settings where the data source has incentives to return biased answers intentionally due to the conflict of interest between the user and the data source. We propose efficient algorithms that reformulate input queries to increase the amount of relevant information in the returned results over biased data sources.

VLDB Workshop Reference Format:
Nischal Aryal, Arash Termehchy, and Marianne Winslett. Querying with Conflicts of Interest. VLDB 2026 Workshop: DASHSys: Systems for Data-centric Agents with Human-in-the-loop.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/OSU-IDEA-Lab/Querying-With-Conflicts.

1 INTRODUCTION

Large-scale data sources, e.g., online databases and LLM-powered AI agents, are widely adopted across almost all sectors of our society, and their influence on users' decision-making continues to grow. Due to financial, social, and political factors, the incentives of data sources and their users are often not perfectly aligned. For example, companies are often said to rework their recommendation and ranking models to promote their own products and undermine those of their competitors [7, 8].

Such incentives to manipulate information are inherent at some data sources, such as shopping websites, which charge sellers a 'referral fee' that is a fraction of the price of items sold on its site [1], so that it profits more when users buy higher-priced versions of a given product. Cost-conscious users who re-sort the results by price may have to wade through many undesirable results before reaching the first good match.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment. ISSN 2150-8097.

Example 1.1. A well-known shopping website (**SW** for short) provides a product-search form with sorting options and checkbox filters on attribute values such as brand, rating, and price range. On the day of this writing, a search for 'headset' returns headphones priced between $24.99 and $39.95 near the top of its Results section, and the remaining items on the page were also headphones. When re-sorted by price, the top hit on the first page is a charging cord, followed by microphones, cables, and one $1.98 headset. Only on the seventh page do headphones start to outnumber other accessories.

Data sources' incentives to manipulate information may also arise from non-financial considerations. To maximize user engagement, some social media sites may promote extreme content, which is not necessarily in anyone else's best interest [9]. More generally, data sources may return information that is biased toward or against controversial topics, political groups, or demographic groups, e.g., giving a low rank to information related to these [6, 9].

In the aforementioned settings, data sources interpret the information in user queries to take actions that do not exactly meet users' preferences and objectives and, in fact, might largely deviate from what users deem as desirable. Current proposals to combat undesirable data source actions usually require the data source to follow and implement certain protocols [9]. For example, researchers have proposed methods to remove bias and ensure fairness for data sources to implement [9]. However, data sources often do not have any incentive to implement these proposals because they conflict with their objectives and business models. Some might also argue that such restrictions limit the freedom to disseminate information or conduct business and may not be ethical or have long-term adverse impacts on our society.

Data sources and users do share at least one interest: a data source would like to satisfy the information needs behind users' queries well enough to keep them engaged. Users can leverage this partial common interest and may combat manipulation by modifying their queries so that the data source interpretations of the submitted queries are those desired by users.

Example 1.2. Consider again the user in Example 1.1 who looks for affordable headphones on SW. If the results are biased towards relatively more expensive products, she can add a price cap to her query. For example, when we reissued the SW headset query with a $19 maximum price, the *second* result returned cost $21, but the others on the first page cost less than $19. Thus, SW does not take the user's query literally, but they do influence the results.

But two can play at this game. The data source may be aware of the fact that users know about its manipulations, e.g., users have observed undesirable biases in their previous experiences. Thus, it may reason that users aim at guiding its results towards the users' desired results. Since it is not in the data source's best interest to accept the modified query as precisely the user's true information

need, the data source may try to recover the original intent behind the modified query. For instance, it might also use its prior belief about users' intents, e.g., gained from external sources or previous interactions with users, to interpret the true intent behind the modified query. Consequently, users may reason about this process and modify their queries further. The agents may reason and make decisions based on possible strategies of each other iteratively to find the best method to formulate and interpret queries.

Example 1.3. In Example 1.2, given that the user has added a degree of bias in the modified query, the data source may reason that the user has overstated their preference for headphones of a particular price, just to offset the data source's bias. Hence, it might avoid placing such headphones in the top positions, as we saw when SW still ranked a $21 headset second even after we limited the price to $19 in Example 1.2. The user might counteract this by further reducing her submitted maximum price for a headset.

We propose a framework that models the communication of the user and the data source with conflicts of interest using queries that return ranked lists of results. We present algorithms that lead the aforementioned recursive reasoning of the users, e.g., client or middleware agent, and data source to a stable state and return relevant information for the original user query. Our preliminary experiments on real-world datasets indicate that our algorithms scale to large data. The proofs of our theoretical results are in [2].

2 FRAMEWORK

Our framework models the user and the data source as agents with different objectives and uses research in game theory on how agents with misaligned goals establish informative communication [3].

Query Language. Users express queries in languages such as SQL that return ranked lists, e.g., via `ORDER BY CASE`. Our framework also extends to other query languages.

Queries. The user has a true information need, i.e., their **intent** τ and sends a query q. The data source *interprets* the query submitted by the user to obtain its own desired query, i.e., **interpretation** β.

Page 1

id	model	brand	rating	price	best seller
e_1	Tune 770	JBL	4.5	149.95	Y
e_2	Q20i	Soundcore	4.6	39.99	N
e_3	A10 Pro	Sisism	4.7	39.50	N
e_4	Sports	Haoyuyan	3.9	299.99	N

Page 3

id	model	brand	rating	price	best seller
e_{39}	Riff	Skullcandy	4.8	21.99	N

Table 1: Returned results for $\mathbf{q}_{\text{rating}}$ on SW

Example 2.1. Consider a user who queries SW to search for headphones. The user's intent τ might be to view headphones ordered by customer rating, which can be expressed as an SQL query with an `ORDER BY` clause on rating. To express this in SW's real-world interface, the user would first type 'headphones' into the search bar, then sorting by rating ($\mathbf{q}_{\text{rating}}$). As of this writing, when we submit $\mathbf{q}_{\text{rating}}$ to SW, the first page of interpretation β results includes top-16 results representing a variety of brands and price points, as partially shown in Table 1. The results page says that it is showing only products with ratings of 4 and up, though the Haoyuyan brand is rated 3.9. Additionally, a relatively expensive JBL product is promoted to the top. Similarly, a Skullcandy headphone is only returned on page 3 of results, though it is rated 4.8.

Strategies. User's submitted query q is often a reformulation of the user's intent τ, e.g., to hide their intent or to offset data-source bias. The user strategy is a mapping from intents to queries. The data source interpretation is not equivalent to the submitted query. For example, if the data source is biased, given q, it computes a biased query β and returns the result of β on its database instance. The data source strategy is mapping from queries to interpretation.

Example 2.2. Continuing with Example 2.1, to offset SW's bias, we submitted a different query $\mathbf{q'}_{\text{rating}}$ that asked for non-major brands (non-JBL) sorted by rating. This resulted in 4.8 and 4.9-rated products from Skullcandy being listed 12th and 14th on the first page, even though all the other products on that page were only rated 4.7. Similarly, the first item on the third page was a 4.8 Skullcandy headset. However, when we searched for 'headphones' with SW's default search order, no Skullcandy products appeared in the first 10 pages of results. This inspires us to imagine a situation where a data source is biased against products from a brand B and demotes them in its interpretation $\boldsymbol{\beta'}_{\text{rating}}$ of the `ORDER BY` clause, e.g., `ORDER BY CASE WHEN brand = 'B' THEN rating - .2 ELSE rating END DESC`. In such a situation, if the user is quite fond of products from B, a better submitted query for user's intent is $\mathbf{q''}_{\text{rating}}$ `ORDER BY CASE WHEN brand = 'B' THEN .2 + rating ELSE rating END DESC`. This lists all B products first, in rating order, followed by other brands in descending order of rating. In the SW interface, one could approximate this by first using the search bar to query for "headphones", sorting by rating, and then checking the box to view Skullcandy headphones first. When we did this, the first page of interpretation $\boldsymbol{\beta''}_{\text{rating}}$ results were all Skullcandy products, though they were not presented in rating order.

Utility Functions. The utility functions formalize the agents' objectives. We denote the user's utility and the data source's utility by $U^r(\tau, \beta)$ and $U^s(\tau, \beta)$, respectively. Each maps an intent τ and an interpretation β to a real value. When the agents' objectives differ, U^r and U^s may be maximized by different interpretations for the same intent. To model this conflict, we assume the data source's utility includes a *bias function* $b(e)$ that maps each tuple e of the relation schema $\mathcal{R}$ to real values. We assume that both agents know each other's utility functions. Learning utility functions from interaction traces is a separate problem. Researchers have proposed inverse game theory and maximum-likelihood methods for learning the utility functions from traces of past agent interaction [5].

Example 2.3. Continuing the running example, the user prefers the data source's results to match the user's intent. To explain this, let $U^r(\tau, \beta) = -\sum_e \big(\mathrm{r}(e, \tau) - \mathrm{r}(e, \beta)\big)^2$, where $\mathrm{r}(e, .)$ denotes the rank of the tuple e under a query. To explain the data source's objective for demoting brand B and promoting expensive products, let $U^s(\tau, \beta) = -\sum_e \big(\mathrm{r}(e, \tau) - b(e) - \mathrm{r}(e, \beta)\big)^2$, where $b(e) = -\mathbf{1}[\mathrm{brand}(e) = B] + \mathbf{1}[\mathrm{price}(e) \geq 100]$. Then U^s is maximized by ranking tuples in increasing order of $\mathrm{r}(e, \tau) - b(e)$.

Utility of Strategies. The data source does not know the user's intent and only observes the query q. However, we assume the

data source knows the prior over the user's intent based on past interaction. Given a submitted query q, the data source updates its belief using Bayes' rule [3], obtaining the posterior $\Phi(\tau \mid q)$ and then computes its expected utility $\mathbb{E}[U^s(\tau, \beta') \mid q]$. The user's expected utility $\mathbb{E}[U^r(\tau, \beta) \mid q]$ is over the data source's strategy.

Equilibrium. An equilibrium is a stable state in which neither agent can improve their utility by changing strategy. In equilibrium, the user finds the query q that (partially) satisfies the user's objective, $q = \arg\max_{q'} \mathbb{E}[U^r(\tau, \beta) \mid q']$. The data source returns results based on an interpretation β that (partially) satisfies its objectives, $\beta = \arg\max_{\beta'} \mathbb{E}[U^s(\tau, \beta') \mid q]$. An equilibrium is influential if the data source's results change based on the queries, and users can combat manipulation by sending appropriate queries.

3 FINDING INFLUENTIAL QUERIES

The data source may steer results toward sponsored, high-margin, or otherwise biased tuples. As a result, directly expressing an intent may yield a ranked list dominated by promoted alternatives. In this section, we first prove general conditions for finding a query that leads to an influential equilibrium, i.e., an influential query, and then propose an efficient algorithm.

Problem. *Given the interaction setting $(\mathcal{R}, \tau, U^r, U^s)$, find a query that leads to an influential equilibrium*

Theorem 3.1. *The interaction $(\mathcal{R}, \tau, U^r, U^s)$ has an influential equilibrium if and only if there are set-equivalent intents $\tau \neq \tau'$ and interpretations $\beta \neq \beta'$ such that, for both utility functions $U^t \in \{U^r, U^s\}$, either $U^t(\tau, \beta) \geq U^t(\tau, \beta')$ and $U^t(\tau', \beta') \geq U^t(\tau', \beta)$, or $U^t(\tau, \beta') \geq U^t(\tau, \beta)$ and $U^t(\tau', \beta) \geq U^t(\tau', \beta')$.*

Theorem 3.1 gives conditions for the existence of influential queries. Here, two intents over schema $\mathcal{R}$ are *set-equivalent* if they return the same set of answers over all instances of $\mathcal{R}$. However, applying the theorem to find influential queries requires searching over intents and interpretations, which is inefficient for expressive query languages. We therefore focus on utility functions with additional structure. Due to space constraints, we present our results for the quadratic utility functions of Example 2.3, which yields an efficient algorithm. Extensions to other general classes of utility functions are discussed in [2].

Relative rank constraints. A user aware of biased rankings may need to communicate not only that a tuple e is preferred to another tuple e' in their intent, but that this preference is very strong. Formally, for a result $\tau(I)$, let $\mathrm{r}(e, \tau(I))$ denote the rank of tuple e, where smaller ranks are better. Given $e, e' \in \tau(I)$ and $\delta \geq 1$, a *relative rank* constraint is $\mathrm{r}(e', \tau(I)) - \mathrm{r}(e, \tau(I)) \geq \delta$. It says that e is ranked at least δ positions above e' in the user's intent.

Possible intent ranks. As discussed in Section 2, the data source does not know the user's intent τ, and therefore maintains a prior over possible intents. Specifically, it maintains a prior over the *ranks* of tuples in $\tau(I)$. Concretely, for each tuple $e \in \tau(I)$, we assume the rank $\mathrm{r}(e, \tau(I))$ is a discrete random variable supported on $[z] = \{1, 2, \ldots, z\}$ and is uniformly distributed on $[z]$.

Theorem 3.2. *For an ordered tuple pair (e, e'), let $b = b(e) - b(e')$, and define $g(z, \delta) = \frac{-\delta^3 + 3\delta^2 z + \delta z^2 + \delta + z^2 - z}{3(z^2 - z + \delta(2z+1) - \delta^2)}$. Let $\delta^\star(e, e')$ be the smallest integer $\delta \in \{1, \ldots, z-1\}$ satisfying $g(z, \delta) - 1 < b \leq g(z, \delta)$, and set $\delta^\star(e, e') = \perp$ if no such integer exists. If $\delta^\star(e, e') \neq \perp$, then the relative-rank constraint is $\mathrm{r}(e', \tau(I)) - \mathrm{r}(e, \tau(I)) \geq \delta^\star(e, e')$.*

Algorithm 1 Finding Influential Queries

Input: interaction setting $(\mathcal{R}, \tau, U^r, U^s)$
Output: query q^δ
$\mathcal{V} \leftarrow \emptyset$
for each ordered pair (e, e') in the rank domain **do**
  $\Delta b \leftarrow b(e) - b(e')$ ▷ Theorem 3.2
  Find the smallest $\delta^\star \in \{1, \ldots, z-1\}$ such that $g(z, \delta^\star) \geq \Delta b$
  if $\delta^\star = \perp$ or $g(z, \delta^\star) - 1 \geq \Delta b$ **then**
    continue
  if $\mathrm{r}(e', \tau) - \mathrm{r}(e, \tau) \geq \delta^\star$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathrm{r}(e', \tau) - \mathrm{r}(e, \tau) \geq \delta^\star\}$
  else
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathrm{r}(e, \tau) - \mathrm{r}(e', \tau) \geq 1 - \delta^\star\}$ ▷ Lemma ??
return $q^\delta \leftarrow \bigwedge_{\ell \in \mathcal{V}} \ell$

Theorem 3.2 provides a closed-form solution to find the relative rank constraint between tuples. Informally, the theorem says that the data source's larger bias in favor of e' requires the user to provide stronger evidence that e truly outranks e' in the intent, i.e., a larger $\delta^\star$. The following result follows from the theorem.

Proposition 3.3. *If there exists a pair e, e' such that $\delta^\star(e, e') \neq \perp$, then the interaction has an influential equilibrium.*

Definition 3.4. Given an intent τ over an instance I of schema $\mathcal{R}$, a **δ-query** q^δ is any conjunction of relative rank constraints of the form $\mathrm{r}(e', \tau(I)) - \mathrm{r}(e, \tau(I)) \geq \delta$, where $\delta \in \mathbb{Z}$. Equivalently, for some set $\mathcal{V} \subseteq \tau(I) \times \tau(I) \times \mathbb{Z}$,

$$q^\delta = \bigwedge_{(e, e', \delta) \in \mathcal{V}} \big(\mathrm{r}(e', \tau(I)) - \mathrm{r}(e, \tau(I)) \geq \delta\big).$$

Algorithm. Algorithm 1 constructs a query q^δ that is satisfied by the input intent τ. Because the user may not know the exact instance I, the algorithm operates over the rank domain induced by the schema and the intent's ranking attributes. For each ordered tuple pair, it computes $\delta^\star$ using Theorem 3.2. The final query is the conjunction of the resulting pairwise constraints. Let $m = \prod_{i=1}^{l} |\mathrm{dom}(A_i)|$ denote the size of the domain induced by the ranking attributes $A_1, \ldots, A_l$. In the worst case, when τ induces a total order, the algorithm examines $m(m-1) = O(m^2)$ ordered pairs. For each pair, it finds $\delta^\star$ by scanning $\delta \in \{1, \ldots, z-1\}$, which takes $O(z)$ time. Therefore, the overall time complexity is $O(m^2 z)$.

Example 3.5. Suppose given a query over `Headphones` instance I, Algorithm 1 outputs the query $q^\delta = \big(\mathrm{r}(e_2, \tau(I)) - \mathrm{r}(e_1, \tau(I)) \geq 1\big) \wedge \big(\mathrm{r}(e_4, \tau(I)) - \mathrm{r}(e_3, \tau(I)) \geq 1\big)$, This conjunction requires e_1 to be ranked above e_2 and e_3 to be ranked above e_4, but imposes no constraint between the pairs $\{e_1, e_2\}$ and $\{e_3, e_4\}$. Hence, q^δ is consistent with multiple ranked lists. For instance, $q_1^\delta(I) : e_1 \prec e_2 \prec e_3 \prec e_4$, $q_2^\delta(I) : e_3 \prec e_4 \prec e_1 \prec e_2$ both satisfy q^δ.

Limitations on query language. Supporting queries like q^δ requires a language that can express constraint-based ranking specifications or a data source that accepts a union of ranking queries. As discussed above, q^δ denotes the set of rankings. This set determines whether q^δ can influence the data source's interpretation. If it is empty, q^δ is uninfluential. If it covers the entire space of queries, e.g., all permutations of tuples in the user's intent, then

Table 2: Runtime by number of attributes. Runtime is in 10^{-3} seconds.

Dataset	# Attributes		
	1	2	3
Amazon	0.042	1.178	1473.642
Flights	0.023	0.365	60.636
PriceRunner	0.045	3.042	396.716

Table 3: Runtime and utility gain across bucket sizes. Runtime is in 10^{-3} seconds.

Dataset	Metric	# Buckets								
		2	5	12	27	62	140	315	710	1065
Amazon	Time	5.827	8.908	10.478	10.867	11.151	11.899	10.896	12.269	13.589
	Utility Gain	0.00	0.00	5006.90	5552.69	5842.18	5831.69	6318.94	6369.49	6383.91
Flights	Time	1.545	2.509	2.704	2.988	3.099	3.122	3.375	3.978	4.491
	Utility Gain	0.00	0.00	34777.80	39716.25	40732.95	40637.73	41025.24	41318.98	41576.12
PriceRunner	Time	1.633	2.432	2.832	3.077	3.078	4.883	3.608	4.419	4.809
	Utility Gain	0.00	5535.63	5692.93	5709.11	5722.53	5788.53	5788.53	5787.35	5787.35

q^δ is again uninfluential. Only when q^δ rules out some rankings but not all can it convey nontrivial information. Then users can submit q^δ, and it leads the interaction to an influential equilibrium (Proposition 3.3). However, our current query language (Section 2) is focused on single SQL queries whose ranking is determined by an `ORDER BY` clause. Such queries necessarily produce a single-ranked list. To bridge this gap, we can randomly select one ranking from the non-trivial set and submit its SQL encoding. Extending our methods to other query languages is our ongoing work.

4 EMPIRICAL STUDY

4.1 Experiment Setup

Datasets. We evaluate our method on three real-world datasets of varying sizes, domains, and conflict-of-interest scenarios.

Amazon: This dataset contains over 1.4M product listings with 11 attributes from Amazon [4]. We test for conflict of interest based on attributes: rating, sales in the last month, and price.

PriceRunner: The dataset is collected from PriceRunner, a popular product comparison platform [2]. It includes 35k product offers with 7 attributes provided by different sellers. We test for conflict of interest based on: seller, product model, and product category.

Flights: The dataset was scraped from EaseMyTrip, a flight booking service provider [2]. It contains 300k flight bookings and 12 attributes. We test for conflict of interest based on attributes: airlines, days from trip, and price.

Generating queries. For each dataset, we generate queries using the conflict of interest attributes described above as candidate ranking attributes. Each query has the form `SELECT A_1, ..., A_l WHERE Sel(q) ORDER BY A_1, ..., A_l`, where $l \in \{1, 2, 3\}$. Varying l lets us study how our algorithm scales as the number of attributes increases. The attributes $A_1, \ldots, A_l$ are used in the `ORDER BY` clause, and we include the same attributes in the `SELECT` clause for simplicity. The selection condition `Sel(q)` is a random conjunction of up to three predicates over non-key attributes. Numeric ranking attributes are ordered in descending order, while categorical attributes use their order in the original dataset.

Bucketization. The complexity of Algorithm 1 depends on the size of the rank domain induced by the intent's ranking attributes, which can be large when attributes have high-cardinality domains. Following standard practice [6], we therefore bucketize attributes with a large domain, e.g., price. We provide additional details in [2]. Since the selection of bucketization may affect the user's utility and the running time of our algorithm, we also evaluate this effect.

Baseline and metrics. We compare Algorithm 1 (IQ) against the baseline that directly submits the user's intent without reformulation. With quadratic user utility, $U^r(\tau, \beta) = -\sum_{e \in \tau(I)} (\mathrm{r}(e, \beta(I)) - \mathrm{r}(e, \tau(I)))^2$. We report utility gain as $U^r(\tau, \beta_{\mathrm{IQ}}) - U^r(\tau, \beta_{\mathrm{base}})$, so positive gain means that IQ reduces squared rank distortion relative to the original query. To model data-source bias consistently across datasets, we randomly sample normalized bias values from $[0, 1]$ and scale them to the domain of attributes in the query. All our reported results are averaged over five runs.

4.2 Experiment Results

Impact of number of attributes on execution time. We study how the rank-domain size affects runtime by varying the number of ranking attributes from 1 to 3. Table 2 shows that the algorithm is extremely fast for one and two attributes, with runtimes below 3.1×10^{-3} seconds. Runtime increases for three attributes, as expected from the larger search space, but remains practical: even the largest case, Amazon, finishes in under 1.5 seconds.

Impact of bucketization on execution time and utility. We evaluate bucketization on the highest-cardinality attribute in each dataset. We use dyadic refinement, starting from two contiguous buckets and repeatedly splitting buckets to obtain larger bucket counts. For each level, we run Algorithm 1 (IQ) and evaluate the resulting query on the full unbucketized domain, with a 10-minute budget per level. Table 3 shows that IQ scales smoothly with bucket size. Runtime remains below 14×10^{-3} seconds across all datasets, while utility gain rises quickly at small to moderate bucket sizes and then plateaus. Thus, moderate bucketization captures most of the utility gain with little runtime overhead.

REFERENCES

[1] Amazon. 2025. Amazon selling fees. https://sell.amazon.com/pricing

[2] Nischal Aryal, Arash Termehchy, and Marianne Winslett. 2026. Querying with Conflicts of Interest. arXiv:2603.05704 [cs.DB] https://arxiv.org/abs/2603.05704

[3] Vincent P. Crawford and Joel Sobel. 1982. Strategic Information Transmission. *Econometrica* 50, 6 (1982), 1431–1451. http://www.jstor.org/stable/1913390

[4] Aaron Frias. 2025. Amazon Products Dataset. Kaggle. https://www.kaggle.com/datasets/aaronfriasr/amazon-products-dataset/data.

[5] Volodymyr Kuleshov and Okke Schrijvers. 2015. Inverse Game Theory: Learning Utilities innbsp;Succinct Games. In *Web and Internet Economics: 11th International Conference, WINE 2015.* https://doi.org/10.1007/978-3-662-48995-6_30

[6] Jinyang Li, Yuval Moskovitch, and H. V. Jagadish. 2023. Detection of Groups with Biased Representation in Ranking . In *2023 IEEE 39th International Conference on Data Engineering (ICDE).* https://doi.org/10.1109/ICDE55515.2023.00168

[7] Dana Mattioli. 2019. Amazon Changed Search Algorithm in Ways That Boost Its Own Products. *Wall Street Journal* (16 September 2019). https://www.wsj.com/articles/amazon-changed-search-algorithm-in-ways-that-boost-its-own-products-11568645345

[8] Jack Nicas and Keith Collins. 2019. How Apple's apps topped rivals in the App Store it controls. *The New York Times* (9 September 2019). https://www.nytimes.com/interactive/2019/09/09/technology/apple-app-store-competition.html

[9] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2022. Fairness in Ranking, Part II: Learning-to-Rank and Recommender Systems. *ACM Comput. Surv.* 55, 6, Article 117 (dec 2022), 41 pages. https://doi.org/10.1145/3533380