CONCIERGE: Improving Constrained Search Results by Data Melioration

Ido Guy[1] idoguy@acm.org
Tova Milo[2] milo@post.tau.ac.il
Slava Novgorodov[1] snovgorodov@ebay.com
Brit Youngmann[2] brity@mail.tau.ac.il
[1] *eBay Research*
[2] *Tel Aviv University*

ABSTRACT

The problem of finding an item-set of maximal aggregated utility that satisfies a set of constraints is at the cornerstone of many e-commerce applications. Its classical definition assumes that all the information needed to verify the constraints is explicitly given. In practice, however, the data available in e-commerce databases on the items is often partial. Hence, adequately answering constrained search queries requires the completion of this missing information. A common approach to complete missing data is to employ Machine Learning (ML) algorithms. However, ML is naturally error-prone. More accurate data can be obtained by asking the items' sellers to complete missing data. But as the number of items in the repository is huge, asking sellers about all items is prohibitively expensive. CONCIERGE, our presented system, assists the e-commerce platform in identifying a bounded-size set of items whose data should be manually completed, as these items are expected to contribute the most to the constrained search queries in question. We demonstrate the effectiveness of our system on real-world data and scenarios taken from a large e-commerce system by interacting with the VLDB'20 participants who act as both analysts and the sellers.

PVLDB Reference Format:
Ido Guy, Tova Milo, Slava Novgorodov, and Brit Youngmann. CONCIERGE: Improving Constrained Search Results by Data Melioration. *PVLDB*, 13(12): 2865-2868, 2020.
DOI: https://doi.org/10.14778/3415478.3415495

1. INTRODUCTION

The selection of a k-size item-set with the maximal aggregated utility that satisfies a set of constraints is a fundamental problem to many e-commerce applications. As an example, consider a user searching an e-commerce website for shirts. Rather than simply returning the top-k items matching the user's request (according to their utility scores), the platform often takes into account additional requirements. For instance, it might have signed a contract with a particular brand, requiring that the query's result set contains at least one item of this brand. It may also wish to diversify the result set, including at least two other brand names, as well as shirts having different sleeve length or colors [7, 3].

The problem of finding an item-set of maximal aggregated utility that satisfies a set of constraints is often referred to as *Constrained Search* (CS) [3]. Its classical definition in the literature assumes that all the data needed to verify the constraints is explicitly given. In practice, however, the data in e-commerce databases on the items is often lacking. Sellers frequently upload goods in batches and tend to focus, in their provided information, only on the most important attributes (e.g., product name and cost), with additional information given through natural language description and/or an image. Hence adequately answering constrained search queries requires the completion of this missing data.

A common approach to complete missing data is by employing Machine Learning (ML) algorithms [8, 10, 5]. However, ML is naturally error-prone, and previously-reported results indicate that it is hard to achieve 85% precision for a reasonable recall [6]. More accurate and authoritative data can be obtained by asking the sellers to complete missing information. But as the number of items in the database is typically huge, limiting human effort is crucial. To tackle this challenge, we propose a hybrid approach that *harnesses the information derived by common ML modules to reduce the manual effort*, focusing on the potentially most "beneficial" items. Given a set of constrained search queries of interest and a bound on the number of requests from the sellers, CONCIERGE, the system that we present in this work, assists the e-commerce platform in identifying a bounded-size set of items whose data should be manually completed. To this end, it considers the probabilities derived by the ML modules for the missing attribute values. It identifies a candidate set of items that are expected to contribute the most to the constrained search queries (in terms of both constraint satisfaction and utility). This is achieved by employing a dedicated algorithm that we experimentally show to be effective, despite the inherent complexity hardness of the corresponding optimization problem. CONCIERGE then generates data verification questions to the relevant sellers, updating the repository accordingly.

Before presenting CONCIERGE, let us illustrate the problem that we address in this work through a simple example.

Example 1.1. Consider an e-commerce platform where sellers upload women shirts. The items database is depicted in Figure 1. It includes information about the shirts' brand names and sleeve lengths. Some of the values were provided by the sellers. Missing values were derived using ML algo-

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415495

rithms. Next to each given/derived value, we also show (in the parenthesis) the alternative possible values, along with their corresponding probabilities, as determined by the ML module. W.l.o.g. assume that the e-commerce platform chooses the value with highest probability as the attribute value. This may, or may not, match the actual ground truth value (marked here in red). We examine two search queries: "women shirts" (q_1), and "women sport shirts" (q_2). Let k, the number of items to be returned, be 3. The utility scores of the items w.r.t the queries are also depicted in Figure 1. During a season change, the platform imposes a constraint requiring the queries' results to include items having different sleeve lengths. Also, assume that the platform has signed a contract with `Versace`, requiring all result sets to include at least one `Versace` item. To ensure coverage of the brands, another constraint was imposed, demanding to include items having different brands.

The platform's result for q_1 is the set S_1=$\{1, 7, 8\}$ (with utility of 1.8), and its result for q_2 is S_2=$\{4, 5, 8\}$ (with utility of 1.8). Note, however, that S_1, in fact, does not satisfy the constraints w.r.t the ground truth, and the result for q_2 could be improved in terms of utility, if the ground truth was known. Turning to the seller of item 7 to complete missing data would improve both queries' results: For q_1 the result becomes S'_1=$\{1, 2, 7\}$ (with utility of 2.6), and for q_2 the result alters to S'_2=$\{4, 5, 7\}$ (with utility of 2.7). Moreover, both S'_1 and S'_2 also satisfy the constraints w.r.t. the ground truth. Turning to the seller of item 2, on the other hand, would improve the result only for q_1, altering it to S'_1. In contrast, completing missing values on items 8 or 9 is redundant: Item 8 is irrelevant to the queries (it has low utility scores), and item 9 has alternative items with higher probabilities which can meet the constraints' requirements (item 7 is more likely to have the brand `Versace`).

CONCIERGE uses the probabilities derived by the ML modules to choose a bounded-size set of items that is expected to improve both the utility and the probability of satisfying the constraints, for both queries. In our example, assume that we wish to bound the number of information requests to 3. CONCIERGE would turn to the sellers of items 2, 6, and 7, resulting with the optimal (w.r.t the ground truth) solutions for both queries: S'_1 and S'_2 (see formal definition and further details in Section 2).

CONCIERGE handles multiple queries simultaneously by supporting two commonly used aggregation strategies: Average and Least Misery (LM). With the average aggregation function, the system selects a bounded-size item-set that is expected to maximize the average contribution to the examined queries. Using the LM function, it selects an item-set that maximizes the minimum contribution for each of the queries. Via CONCIERGE's dedicated UI, the user can: (1) select the search queries of interest and their imposed constraints; (2) limit the overall number of requests from sellers, and (3) define the aggregation strategy.

While our exposition on the features of our system focuses on an e-commerce use case, we note that CONCIERGE is a general-purpose system applicable to data melioration in general search applications, including search engines (e.g., Google) and online media sites (e.g., Netflix).

We demonstrate the operation of CONCIERGE over real-world e-commerce data. Our demonstration illustrates a real-life scenario where a data analyst attempts to improve the results of the most popular (constrained) search queries

Item	Utility(q_1)	Utility(q_2)
1	0.9	0.001
2	0.8	0.001
3	0.7	0.001
4	0.001	0.9
5	0.001	0.9
6	0.001	0.9
7	0.9	0.9
8	0.001	0.001
9	0.6	0.8
10	0.4	0.001

Figure 1: **Example database which include information about the items brand name (where the values are Gucci (g), Versace (v) and Nike (n)), and sleeve lengths (where the values are long (l) and short (s)). On the bottom are the items' utility scores w.r.t. two queries.**

on the website. The audience will play the role of data analysts, selecting the queries of interest. Then, the participants will explore the items chosen by CONCIERGE to be cleaned and, also, playing the role of the sellers, will complete missing values (by examining the items' descriptions and pictures). Last, the audience will examine the improvement in the affected search queries.

Related work. Our work is closely related to a line of work studying different variants of the CS problem, proposing efficient algorithms for solving them [7, 9, 3]. While we establish the connection between CS and the optimization problem that we study in this work (showing our problem to be harder), we emphasize that our goal is different. Instead of finding the optimal solution for the search queries, we aim to improve their results, via data melioration.

Multiple data cleansing tools combine both human and ML [8], typically using domain experts to generate adequate labeled data for supervised learning, while minimizing human effort [10, 5]. Our work complements these previous efforts by leveraging the probabilities obtained by the ML algorithms, to identify which items should be manually cleaned. CONCIERGE can be used to optimize the cleaning process of a database, as well as to assist in its ongoing maintenance - whenever a new constraint is imposed, CONCIERGE can take over to efficiently identify what missing information may improve the queries' results.

Query evaluation over probabilistic databases is a well-studied problem [4, 2]. In this work, we adapt tools developed for this task (e.g., the *possible worlds semantics* of [4]), showing they are useful for data cleansing as well.

2. TECHNICAL BACKGROUND

We first present our data model, then formally define the Probabilistic Constrained Search (PCS) problem, and pro-

vide two extensions for handling multiple queries. Last, we briefly present our algorithms. Further details can be found in our online appendix [1].

2.1 Data Model

We are given with a set $\mathcal{I}$ of n items, each associated with a set of attributes, and for each attribute a set of value-probability pairs, capturing the probability of the attribute to have each value. Given a search query q, each item $i \in \mathcal{I}$ is also associated with a utility score, denoted as $u_q(i)$, which measures the expected satisfaction of a consumer to i w.r.t. q. This score is a function of the relevance score of an item to q, and some static scores associated with it (e.g., based on customers' reviews). Following [7], the utility of an item-set I w.r.t. q denoted as $u_q(I)$, is defined as the sum of utility scores of the items in I.

A constraint is defined over an attribute a, and an item-set of size k. We consider two simple types of constraints, which can capture a large range of constraints previously studied [7, 9, 3]: *count* and *coverage* constraints. A count constraint defines upper and lower bounds on the number of items *having a specific value for the attribute a*. A coverage constraint defines upper and lower bounds on the number of *different values for the attribute a*. Given a set of items I and a set C of constraints, let $Pr_C[I]$ denote the probability that I satisfies all constraint in C. We compute $Pr_C[I]$ using the chain-rule, addressing the constraints dependencies.

Example 2.1. Continuing with Example 1.1, c_1 is a coverage constraint requiring the result sets to include items having at least two different sleeve lengths; c_2 is a count constraint requiring the results to include at least one `Versace` item; c_3 is a coverage constraint requiring the result sets to include items having at least two different brands.

2.2 Problem Formulation

We can improve the result of a constrained search query q in two manners: increase the overall utility or satisfy the constraints with higher probability (possibly at the cost of reducing utility). The optimal result for q is a k-size item-set that is *most likely to satisfy the constraints while maximizing utility*. This set may be different than the one currently returned by the platform, which may have a lower utility or a lower probability of satisfying the constraints than this set. Intuitively, we would like to make sure that the information on the items in the optimal set is clean and complete, so that it will be possible to include them in the result set of q (as these items have the highest potential to improve it). For that, we define PCS as follows: Find a set I s.t:

$$I = argmax_{|I'|\leq k, I' \subseteq \mathcal{I}} Pr_C[I] \cdot u_q(I) \quad (1)$$

In our problem definition, we set the size of the selected item-set to be k - the number of items in a query result set. In case the bound on the number of requests from the sellers, denoted as k', is $<k$, we will approach only to the sellers of the top k' items with the highest utility scores. In case $k'>k$, we may repeat the cleaning process $\lceil \frac{k'}{k} \rceil$ times.

Example 2.2. Recall that the result of q_1 (based on the predicted values) is $S_1=\{1,7,8\}$, with $u_{q_1}(S_1)=1.8$ and $Pr_C[S_1]=0.55$. According to Equation 1, the item-set that should be manually cleaned is $S'_1=\{1,2,7\}$ with $u_{q_1}(S_1)=2.6$ and $Pr_C[S'_1]=0.44$. S'_1 includes the top 3 items with the highest utility scores w.r.t. q_1. Completing missing data on these items assists the platform to include them in the result set of q_1 (after the cleaning process, the result of q_1 becomes S'_1), improving it in terms of both utility and probability of satisfying the constraints (as S_1 does not satisfy c_1).

Next, we extend PCS to handle multiple queries. One approach to do so is to handle each query separately. However, to limit the overall number of requests from sellers, CONCIERGE handles all examined queries simultaneously. We consider two aggregation strategies: *Average* and *Least Misery* (LM). Let $Q=\{q_1,...,q_m\}$ be a set of m constrained search queries. For simplicity, we assume that the same constraints are imposed over all queries. In our first problem definition, called AVG-PCS, the goal is to find a k-size set of items that maximizes the average contribution across all queries. Formally, find a set I s.t:

$$I = argmax_{|I'|\leq k, I' \subseteq I} \frac{\sum_{q_i \in Q} Pr_C[I'] \cdot u_{q_i}(I')}{m} \quad (2)$$

In our second problem definition, called LM-PCS, the goal is to maximize the minimum contribution for each query. Formally, find a set I s.t:

$$I = argmax_{|I'|\leq k, I' \subseteq \mathcal{I}} \min_{i \in [1,...,m]} Pr_C[I'] \cdot u_{q_i}(I') \quad (3)$$

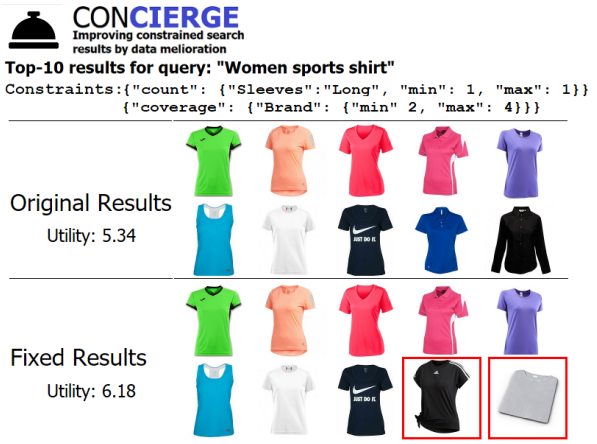
Example 2.3. Following the average policy the selected item-set is $S_{AVG}=\{2,6,7\}$. Informally, item 7 is relevant for both queries, and items 2 and 6 are relevant only for q_1 and q_2, resp. Hence, completing missing data on these items may improve the results of both queries. Following the LM policy the selected set is $S_{LM}=\{6,7,9\}$. Intuitively, as there are fewer items that are relevant for q_2 than for q_1, optimizing the result of q_2 is more challenging. Thus, this set includes more items that are relevant for q_2 (all items in S_{LM}) than items that are relevant for q_1 (only items 7 and 9).

Hardness Results. We note that computing the probability a k-size item-set satisfies a constraint is exponential in k. We, therefore, estimate this probability using the *possible worlds semantics* of [4], showing that it can be estimated up to a constant factor in $O(k)$. As both AVG-PCS and LM-PCS naturally generalize PCS, we provide our hardness results w.r.t. PCS. We also discuss the hardness of CS, which is a restricted variant of PCS, assuming all probabilities are 0 or 1. While in the simple setting where all constraints are defined on a single attribute, there exists an optimal PTIME algorithm for CS [7], we show PCS to be NP-hard, even for this restricted case. For the general case, we show that both PCS and CS are NP-hard, and cannot be approximated to a constant factor in PTIME. We prove this bound to hold for PCS, even if we know which k-size item-set satisfies the constraints with the highest probability, has maximal utility.

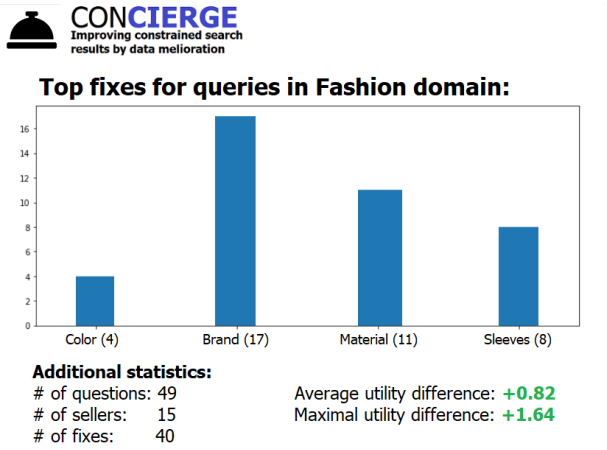
2.3 Algorithms

Since PCS cannot be approximated to a constant factor in PTIME, we provide an efficient best-effort algorithm, which we experimentally show to be highly effective. Our algorithm employs two procedures (described below). We first describe how our algorithm operates over a single query, then explain how to extend it to handle multiple queries.

Greedy initialization. This procedure iterates over the constraints. While count constraints require to select items having a specific value for a given attribute a, coverage constraints require to select items with no particular values for a. We, therefore, first iterate over the count constraints. In each iteration, we select new items to satisfy the currently-examined constraint, while also considering the items selected so far. The main challenge here is to carefully account



(a) **Results of a single constrained search queries**



(b) **Statistics for all examined constrained search queries**

Figure 2: **CONCIERGE UI.**

for the items selected so far, ensuring that the solution still satisfies previously-examined constraints (i.e., does not exceed their upper bounds), while also satisfying the current constraint with as highest probability as possible.

Improvement via local search. This procedure starts with the solution returned by the previous step. It iteratively moves to a neighbor solution by replacing some item(s) with different item(s) having higher utility scores. The main challenge here is to define sufficient conditions on the new item(s) to be added, ensuring improvement. Namely, to ensure that if the probability of satisfying the constraints decreases, the utility will be high enough.

Supporting multiple queries. For AVG-PCS, we employ our algorithms while considering the average utility scores of the items. For LM-PCS, we compute a solution by running our algorithms for each query $q_i \in Q$. Let S_{q_i} denote the solution for the i-th query. We then estimate, for each $q_j \in Q$, the value of $Pr_C[S_{q_i}] \cdot u_{q_j}(S_{q_i})$, and return the set in which its minimum value w.r.t. all queries is maximal.

3. SYSTEM AND DEMONSTRATION

We have implemented CONCIERGE using Python and Flask. The user interacts with the system using a dedicated UI, depicted in Figure 2 and detailed below.

We demonstrate the operation of CONCIERGE over real-world e-commerce data. The audience will play the role of both data analysts, attempting to improve the results of constrained search queries, as well as of the items' sellers, requested to complete missing attribute values.

We begin by asking the audience to select (using the system Input Builder, which is omitted from presentation for space constraints): (1) the search queries of interest, and their corresponding constraints (by examining the query log of a popular e-commerce company); (2) a bound on the number of requests from sellers, and (3) the aggregation policy. After receiving the input, CONCIERGE executes the main algorithm. The output of this algorithm is then used to generate data cleaning tasks that are sent to the relevant sellers. The cleaning tasks can be performed in the dedicated UI, or exported as JSON files and loaded to various crowd platforms. The audience will then be asked to play the role of the corresponding sellers and to complete missing values by examining the items' descriptions and pictures.

Once the cleaning process is complete, the audience can inspect the improvement of the queries, analyzing the effect by comparing their result-sets before and after the cleaning (using the underlying search engine to retrieve the results). For example, Figure 2a depicts the results before and after the cleaning process for the query "women sport shirt". One can see that two items were replaced, and consequently, the overall utility was increased. Note that the last item in the original result set is a long-sleeved button-up shirt. It was replaced with a long-sleeved sport shirt that appears folded in its picture and hence was not detected by the ML module. In addition, the audience can examine statistics describing the results for all affected queries (as presented in Figure 2b). These statistics include the number of performed fixes, distributions of the affected attributes, average utility gain per query, and other useful details.

Last, the audience will be allowed to look "under the hood", examining the effectiveness and efficiency of our algorithms, compared with the naïve approach, that computes the optimal solution using an exhaustive search. For this part of the demonstration, we will use growing fragments of the underlying database, showing the limitations of the naïve approach to scale.

Acknowledgment. This work has been partially funded by the Israel Science Foundation, the Binational US-Israel Science Foundation, and the Tel Aviv University Data Science center.

4. REFERENCES

[1] Online appendix. https://bit.ly/2QiaBHE, 2020.
[2] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. Technical report, Stanford, 2005.
[3] L. E. Celis, D. Straszak, and N. K. Vishnoi. Ranking with fairness constraints. *arXiv preprint arXiv:1704.06840*, 2017.
[4] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 2007.
[5] C.-J. Ho, S. Jabbari, and J. W. Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML*, 2013.
[6] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman. Matching unstructured product offers to structured product specifications. In *KDD*, 2011.
[7] J. Stoyanovich, K. Yang, and H. Jagadish. Online set selection with fairness and diversity constraints. In *EDBT*, 2018.
[8] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *Proc. VLDB Endow.*, 7(13):1529–1540, 2014.
[9] T. Wu, L. Chen, P. Hui, C. J. Zhang, and W. Li. Hear the whole story: Towards the diversity of opinion in crowdsourcing markets. *Proc. VLDB Endow.*, 8(5):485–496, 2015.
[10] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, 2015.