Favorite+: Favorite Tuples Extraction via Regret Minimization

Min Xie
Shenzhen Institute of Computing Sciences, Shenzhen University, China
xiemin@sics.ac.cn

Yang Liu
School of Computer Science and Engineering, Beihang University, China
1844061236@buaa.edu.cn

ABSTRACT

When faced with a database containing millions of tuples, a user might be only interested in some of them. In this paper, we study how to help an end user to find the favorite tuples based on the recent advancements in *regret minimization* queries, which guarantees the tuples returned are not far from the user's favorite tuple in the database, without asking the user to scan the entire database.

We consider three types of regret minimization queries: (1) *End-to-end query*: Given an output size k, we directly return a subset of at most k tuples from the database; (2) *Interactive query*: We identify the user's favorite tuple via *user interaction*, where a user might be presented with a few pairs of tuples, and the user is asked to indicate the one s/he favors more from each pair; and (3) *Incremental query*: Analogous to how we use search engines, if the user is not satisfied with the current tuples, we continually return more. We developed a demonstration system, called Favorite+, by supporting the above queries. We demonstrate that the system could help the users to find their favorite tuples in the database efficiently and effectively.

CCS CONCEPTS

• **Information systems** → **Data analytics**.

KEYWORDS

regret minimization; user interaction; data analytics

ACM Reference Format:
Min Xie and Yang Liu. 2022. Favorite+: Favorite Tuples Extraction via Regret Minimization. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557188

1 INTRODUCTION

Nowadays, a database typically contains millions of tuples, but only some of them are needed by the user. To assist the user for finding the tuple s/he is interested in, operators have to be developed. Such operators are useful in various domains, including house buying, car purchase and job search. Consider a used car database, where each car tuple is described by some attributes (e.g., price, year purchased and horse power). Assume that Alice wants to find an inexpensive and new car from the database. In the literature [3, 6, 11], Alice's preference is represented by a monotonic preference function, called a *utility function*, in her mind. Based on this function, each car in the database has a *utility* and the car with the highest utility is the *favorite* car of Alice. There can be many candidate cars in the database (even after filtering some uninteresting cars which do not satisfy the criteria specified by Alice) and thus, Alice might have to trade-off between different attributes (e.g., Alice might be willing to pay more on buying a used car that is recently purchased than buying an old one). This trade-off is individualistic and not known by the system in a complete way. Hence, traditional queries, e.g., the top-k query, cannot be applied in this scenario.

In this paper, we tackle this problem using the regret minimization query. Intuitively, this query quantifies how regretful a user is using a criterion called *"regret raio"*, if s/he is provided with a small set of tuples, but not the favorite tuple in the database, and it aims to make the regret ratio as small as possible, without asking the user to provide the utility function explicitly. We consider three types of regret minimization queries in this paper, namely the *end-to-end* query, the *interactive* query, and the *incremental* query.

The proper choice of queries depends on the application needs and the levels of interaction that a user is willing to provide. To illustrate the trade-offs between different queries, assume that we are helping Alice to find her favorite car in the database. If Alice cannot provide any information on her utility function, she can execute the *end-to-end* query, by providing a maximum output size k. Then, the system selects a set of at most k cars from the database and show them to Alice, guaranteeing the highest utility of the cars returned is not much worse than the utility of Alice's favorite car in the database, regardless of Alice's utility function. Alice can examine the cars in the returned set and find a car that is the closest to her favorite one. Alternatively, although Alice cannot explicitly tell us her utility function (which is difficult to answer), she might be willing to provide us some "hints" on what her utility function might look like (which is easier to provide), via the interactive query. Specifically, the system can present Alice with a few pairs of cars and ask Alice to pick the one she favors more from each pair. The car favored by Alice might differ from other non-favorite cars in some ways, which reflects the trade-off in Alice's mind. With more rounds of interaction, we can implicitly learn more about Alice's utility function and determine her favorite car in the database. This kind of interaction is considered in [6, 7, 9] and naturally appears in our daily life. Moreover, if we have shown Alice some cars but Alice is still not satisfied, the system can run the incremental query, to return k more cars for Alice to consider. This is analogous to how we use search engines. This iterative process continues until Alice is satisfied with the results. In this paper, we automate the above process by supporting all three types of queries, which has many applications such as product recommendation and self-guided shopping.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557188

There are some existing systems [1, 7] which approach the problem based on the regret minimization query. However, they have the following drawbacks: (1) They only support some interactive regret-minimization queries and none of them have considered the end-to-end query and the incremental query, which are also useful in many practical scenarios. (2) For the interactive queries they support, they either asked the user to spend great effort on *sorting* a short list of candidate tuples, or asked the user a lot of questions to determine the desired favorite tuple, which is not practical. (3) Users are not allowed to manipulate the data via the existing systems, e.g., for data upload, attributes selection, preference indication. There are also some machine learning based systems [2, 5] which conduct interactive data exploration with a similar goal. However, their methods cannot provide any guarantee on the result quality, while our methods are able to theoretically guarantee the degree of satisfaction of the users, by providing an upper bound on their regret ratios.

To address these limitations, we develop a novel system called Favorite+, which is powered by our recent advancements in regret minimization queries in [6, 8, 10], and it has the following attractive features. Firstly, it integrates the end-to-end query, the interactive query and the incremental query in the same system, to fit different users' needs. Secondly, we support a novel interactive query inspired by [6], so that we can identify the user's favorite tuple with a small amount of interactions. Thirdly, we allow the users to manipulate the data, by (1) uploading user-defined datasets, (2) selecting the attributes that they are interested in, and (3) indicating their preference on each attribute. Finally, Favorite+ ensures that the tuples returned is not much worse than the user's favorite tuple in the database, since the methods used in our system provides theoretical guarantee on the regret ratios. In this paper, we demonstrate how to use Favorite+ to help users finding their favorite tuples in the database. Our major contributions are summarized:

- We develop a system, Favorite+, for finding the user's favorite tuple in the database, via three types of regret minimization queries (end-to-end, interactive and incremental).
- Favorite+ provides several interfaces and supports novel functionalities, such as attribute selection and preference indication, for users to flexibly manipulate the data.
- We deploy Favorite+ on a used car database for an easy-to-follow system demonstration, although an end user can upload arbitrary user-defined datasets to use our system.

2 SYSTEM ARCHITECTURE

The input to our problem is a set D with n tuples (i.e., $|D| = n$) in a d-dimensional space (i.e., each tuple is described by d attributes).

We denote the i-th dimensional value of a tuple $p \in D$ by $p[i]$ where $i \in [1, d]$. Without loss of generality, we assume that a larger value in each dimension is more preferable to the user.

Same as [1, 6, 7], the user preference can be represented by an unknown *linear utility function*, denoted by f, which is a mapping $f: \mathbb{R}_+^d \to \mathbb{R}_+$. A function f is *linear* if $f(p) = u \cdot p$ where $f(p)$ is the *utility* of p w.r.t. f and, u is a d-dimensional non-negative *utility vector* where $u[i]$ measures the importance of the i-th dimensional value in the user preference. We assume that $\sum_{i=1}^{d} u[i] = 1$ in this paper and we call $\mathcal{U} = \{u \in \mathbb{R}^+ | \ \sum_{i=1}^{d} u[i] = 1\}$ the *utility space*. In particular, a user is interested in finding the tuple in D which maximizes the utility, denoted by $\mathrm{U}_{\max}(D, u) = \max_{p \in D} u \cdot p$. The tuple with the maximum utility is the *favorite* tuple of this user.

Given a set $S \subseteq D$ and a user with utility vector u, the *regret ratio* of this user is defined to be $1 - \frac{\mathrm{U}_{\max}(S,u)}{\mathrm{U}_{\max}(D,u)}$. Intuitively, when the maximum utility of S is closer to the maximum utility of D, the regret ratio is smaller and the user feels more satisfied with S.

2.1 Architecture Overview

The architecture overview of our system, Favorite+, is shown in Figure 1. The users start with data preparation, including data uploading, attribute selection, preference indication. Then, they can choose a proper type of regret minimization query depending on their needs. (1) If the end-to-end query is chosen, users are further asked to input an integer k, indicating the maximum number of tuples they want to see. Then, the system returns at most k tuples from the database, so that the regret ratios of users are guaranteed. (2) If the interactive query is chosen, the system interacts with the users for *rounds*. At each round, we ask the users for their preference among a pair of carefully selected tuples. Based on the feedback, we implicitly learn the user's utility function, until the user's favorite tuple is identified. (3) If the incremental query is chosen, we continually return more tuples to the user, until the user is satisfied with the results returned. In the following, we briefly illustrate the major components of Favorite+ one by one.

2.2 Component Description

Data processing. Users can upload arbitrary datasets to use our system. In order to support the regret minimization query on general datasets, we need to process the data to the designated format. Note that in realty, although each tuple can be described by many attributes, not all of them are equally important to a user in making the decision. For example, price and year purchased are two attributes important to Alice, while horse power is not. Motivated by this, users are allowed to manually select the attributes that they are interested in our system. Moreover, even given the same attribute, the preference of different users can be diverse. For example, when buying a used car, one might want the horse power to be as large as possible, while the other may not want the horse power to be too large, since it is fuel-consuming. Users should be able to indicate their preference on each attribute. Note that other techniques, e.g., skyline computation [4], can also be integrated as pre-processing steps in our system and we omit their details for lack of space.

End-to-end query. Our system supports various algorithms for the end-to-end query [8], which takes an integer k as input, and returns a set S of at most k tuples to the user. Among them, we propose the-state-of-the-art algorithm, denoted by Sphere [10].

Specifically, to construct the set S, Sphere "uniformly" divides the utility space $\mathcal{U}$ into multiple partitions, where the utility vectors in the same partition are similar. We pick a representative utility vector u_s from each partition. For each utility vector u_s picked, we include the tuples with high utilities w.r.t. u_s into S. Intuitively, by doing so, no matter which partition the user's utility vector u lies in, there is a selected representative utility vector, namely u_s, which is in the same partition as u and thus, similar to u. Since u and u_s are similar, the tuples with high utilities w.r.t. u_s will also have high utilities w.r.t. u, so that the user will be satisfied. By returning those

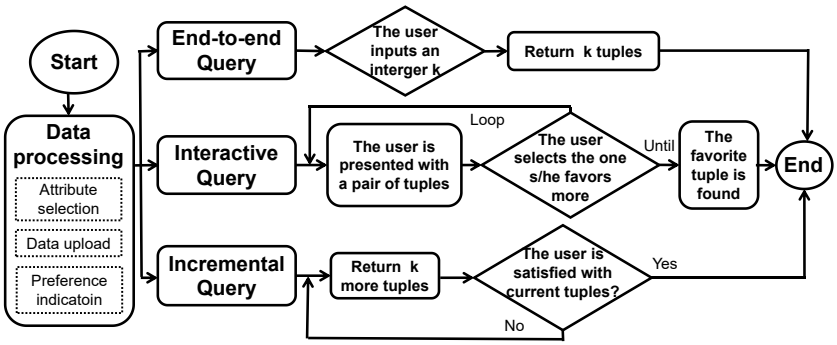

Figure 1: System architecture

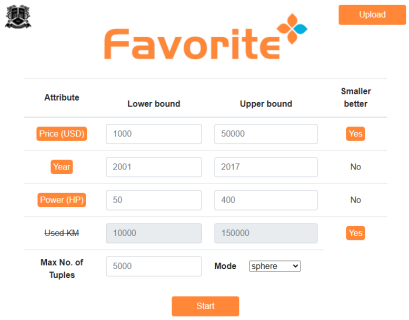

Figure 2: Welcome

tuples in Sphere, the user's regret ratio is guaranteed to be not too large. Formally, Sphere showed the worst-case regret ratio to be $O(k^{-\frac{2}{d-1}})$, regardless of the utility function of users [10]. Other end-to-end algorithms for regret minimization are surveyed in [8].

Interactive query. We present a novel algorithm for the interactive query, by developing a new strategy for selecting the tuple pairs to display to the user, that is not considered in existing systems [1, 7].

Specifically, we maintain a region $\mathcal{R}$ in the utility space which contains the user's true utility vector u. Initially, $\mathcal{R}$ is the entire utility space $\mathcal{U}$. When the user answers more questions, we learn more about his/her preference and $\mathcal{R}$ will be smaller. Note that the user's regret ratio is bounded proportionally to the size of $\mathcal{R}$ [9, 11]. When $\mathcal{R}$ is small enough, we can identify the user's favorite tuple. Assume the user prefers p to q for some tuples p and q in a round. We define a *utility hyperplane*, denoted by $h_{p,q}$, to be the hyperplane passing through the origin O with its unit normal in the same direction as $p - q$. The following lemma shows how we can update $\mathcal{R}$ to be smaller based on $h_{p,q}$, whose proof can be found in [9].

Lemma 2.1. *Given $\mathcal{R}$ and two tuples p and q, if a user prefers p to q, the user's utility vector u must be in $h^+_{p,q} \cap \mathcal{R}$ where $h^+_{p,q}$ denotes the half space above $h_{p,q}$.*

With Lemma 2.1, at each interaction, we select a pair of tuples, namely p and q, such that $h_{p,q}$ is the closest to the geometric center of $\mathcal{R}$. Intuitively, this $h_{p,q}$ partitions $\mathcal{R}$ "the most evenly" and thus, $\mathcal{R}$ is shrunk more rapidly compared with the existing approaches [1, 7]. We denote our algorithm for the interactive query by Parti(tion).

Increment Query. The incremental query has not been considered in regret minimization analysis in the literature, which, however, is also useful in practice. Analogous to how we use search engines, a user can continually obtain more tuples in a incremental query, in case that the user is not satisfied with the current set S of tuples.

Define the *maximum regret ratio* of a set S to be the worst-case regret ratio w.r.t. all utility vectors in $\mathcal{U}$, denoted by $\text{mrr}(S, D) = \max_{u \in \mathcal{U}} 1 - \frac{\text{U}_{\max}(S,u)}{\text{U}_{\max}(D,u)}$. Then the next tuple to be outputted is

$$p^* = \arg\max_{p \in D} \text{mrr}(S, S \cup \{p\}), \tag{1}$$

which can be obtained via Linear Programming (LP). Intuitively, p^* is the tuple in D that makes the users the most regretful and thus, we return it to the user next. In the incremental query, the user is invited to input an integer k, which is the number of additional tuples that s/he wants to see each time. In this case, Equation (1) is executed for k times to obtain k new tuples to return. We denote our algorithm for the incremental query as IncGreedy.

3 SYSTEM DEMONSTRATION

We develop the system called Favorite+, which integrates the data processing and all three types of regret minimization queries. In this section, we demonstrate it on a used car database[1], but users could also upload their own datasets to our system. Recall that Alice wants an inexpensive and new car. We show how to help Alice to find such a car via our system. There are four main interfaces in Favorite+, namely the welcome page, the end-to-end query page, the interactive query page and the incremental query page, in which Alice can manipulate the data and perform the corresponding query. We shot a video[2] to demonstrate Favorite+. The interested readers could find the source code[3] and the demonstration system online[4].

3.1 Data Processing

Figure 2 shows the the welcome page. In this used car database, each car is described by 4 attributes, namely price (in USD), year purchased, power (in HP) and used kilometers. Alice can explicitly select the attributes she is interested in. For example, if Alice do not care about the used kilometers when buying the car, she can explicitly disable this attribute by clicking on the attribute name, as shown in Figure 2. Moreover, Alice can specify her preference on each attribute, e.g., a lower price and smaller used kilometers are more preferable while a more recent purchase and a higher power are more preferable. Similar to the systems in [1, 7], Alice can also specify the acceptable value ranges on all attributes. For example, Alice may specify the price from 1,000 USD to 15,000 USD (i.e., inexpensive cars) and the year purchased from 2015 to 2022 (i.e., new cars). If Alice does not have specific requirement on the attribute values, she can leave them blank to use the default setting. Besides, Alice can also specify the maximum number of cars that she wants to retrieve for selection (e.g., we can use the entire database with 60,000+ cars), and the proper algorithm (i.e., mode) to perform the query. Then, she can click the "Start" button.

After that, Favorite+ processes the dataset for the query stage as follows. Firstly, it retrieves the quantified cars from the database, whose attribute values are in the acceptable ranges specified by

[1] https://www.kaggle.com/orgesleka/used-cars-database
[2] https://youtu.be/kU-5qjtAhyw
[3] https://github.com/SICS-Fundamental-Research-Center/find-your-favorite-car
[4] https://mxieaa.github.io/Favorite

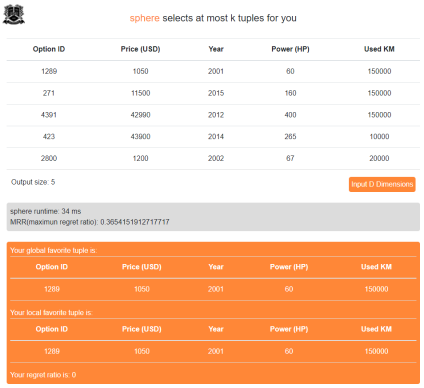

Figure 3: End-to-end query

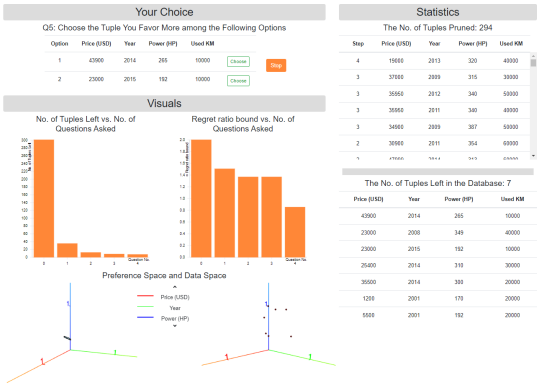

Figure 4: Interactive query

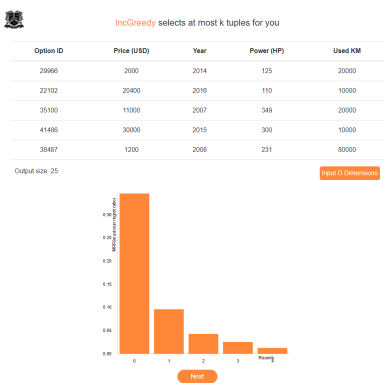

Figure 5: Incremental query

Alice. Secondly, it processes each car, by discarding the disabled attributes. For each remaining attribute, it normalizes the value via min-max normalization if the attribute is not indicated as "smaller better" (i.e., a smaller value is more preferable). Otherwise, the attribute value is processed to be one minus the normalized value.

3.2 Regret Minimization Query

Favorite+ supports three types of regret minimization queries. Below we demonstrate them one by one under different scenarios.

End-to-end query. Favorite+ supports 6 algorithms [8] for the end-to-end query: in addition to the state-of-the-art algorithm Sphere [10], GraphDP, BiSearch, SweepDP, DMM and Cube are implemented for comparison. Among them, GraphDP, BiSearch and SweepDP are 2-dimensional algorithms, restricted to tuples with two attributes. Assume that Alice do not want to provide much feedback, by selecting mode "sphere". She will then be asked to input an integer k, indicating the maximum output size. Assume that Alice enters "5". Then, Favorite+ invokes algorithm Sphere, which returns a set S of at most 5 cars from the database D to Alice, together with the execution time and the maximum regret ratio $\mathrm{mrr}(S, D)$. The resulting interface is shown in Figure 3.

Note that the maximum regret ratio of S is the worst-case regret ratio bound w.r.t. *all* utility functions. The actual regret ratio of Alice can be much smaller. Alice can click the button "Input D Dimensions" at the right of page, to input her utility vector u explicitly, e.g., she can assign weight 40% to "Price" and 60% to "Year". Based on Alice's preference, we can compute the utility of each car, identify her favorite tuples in S and D and compute the actual regret ratio of Alice, as shown in the bottom box in Figure 3. In this particular example, although we just return 5 cars from 60,000+ candidates, her favorite car is indeed returned, giving rise to a 0 regret ratio.

Interactive query. Favorite+ supports 3 algorithms for the interactive query, including our Parti algorithm, and Simplex and Random in [7] for comparison. Assume that Alice agrees to interact with our system. She can enter the interactive interface in Figure 4, by selecting mode "Parti". Specifically, Favorite+ interacts with Alice for rounds and at each round, it displays a pair of cars and asks Alice to pick the one she favors more (the top-left part in Figure 4). According to her feedback, Favorite+ updates the region $\mathcal{R}$ in the utility space and computes the remaining cars in the database. Clearly, with more rounds of interaction, $\mathcal{R}$ is smaller and fewer candidate cars are left. To better understand this process, we plot $\mathcal{R}$ in the preference space and draw each remaining car in the data space in the bottom-left part in Figure 4. Moreover, we provide two histograms "Cars left vs. questions asked" and "Regret ratio bound vs. questions asked" so that the user can visualize the progress. For example in the figure, after Alice answers 5 questions, the number of candidate cars is reduced by 97.6%. The cars pruned and cars remained after each round are shown in the right part. In case that Alice is satisfied the current cars, she can click the "Stop" button to stop immediately.

Incremental query. Alice could also opt to perform the incremental query by selecting mode "IncGreedy" if she wants continuous results. In this case, Alice is asked to be further input an integer k, representing the number of additional cars she wants to see each time. After that, Alice will be presented the first set S of k cars from the database D. Same as the end-to-end query, Alice could also input her utility vector u explicitly, to compare her favorite tuples in S and D. If she is not satisfied the current result S, she can click the "Next" button at the bottom of the page, and obtain k more cars from the database. This process continues until Alice finds her desired car in the results or the database is exhaustive. Note that in general, with more cars returned, Alice's regret ratio will be smaller. For the ease of visualization, we plot a histogram "Maximum regret ratio vs. the number of rounds" in the middle of the page, so that end users can clearly visualize the reduction on regret ratios, e.g., to determine whether they should continue to click "Next" or not.

4 CONCLUSION

In this paper, we show how to find the user's favorite tuples in a large database based on the recent techniques proposed for regret minimization queries. We develop an easy-to-use system, Favorite+, which supports three types of regret minimization queries on general datasets. Users are further allowed to manipulate the data to fit individualistic needs. We demonstrated Favorite+ on a used car database, showing our usefulness and effectiveness.

ACKNOWLEDGMENTS

The research is supported by Longhua Science and Technology Innovation Bureau LHKJCXJCYJ202003 and Guangdong Basic and Applied Basic Research Foundation 2022A1515010120.

REFERENCES

[1] Chen C., J. Zheng, W. Yan, and M. Wang. 2020. IDEAL: IDEntifying the User's IdeAL Tuple via Sorting in the Database. In *CIKM*.

[2] Y. Diao, K. Dimitriadou, Z. Li, W. Liu, O. Papaemmanouil, K. Peng, and L. Peng. 2015. AIDE: an active learning-based approach for interactive data exploration. *VLDB* (2015).

[3] D. Nanongkai, A.D. Sarma, A. Lall, R.J. Lipton, and J. Xu. 2010. Regret-Minimizing Representative Databases. In *VLDB*.

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger. 2005. Progressive skyline computation in database systems. In *TODS*.

[5] X. Qin, C. Chai, Y. Luo, and N. Tangand G. Li. 2020. Interactively Discovering and Ranking Desired Tuples without Writing SQL Queries. In *SIGMOD*.

[6] W. Wang, R. C.-W. Wong, and M. Xie. 2021. Interactive Search for one of the Top-k. In *SIGMOD*.

[7] M. Xie, T. Chen, and R. C.-W. Wong. 2019. FindYourFavorite: An Interactive System for Finding the User's Favorite Tuple in the Database. In *SIGMOD*.

[8] M. Xie, R. Wong, and A. Lall. 2020. An Experimental Survey of Regret Minimization Query and Variants: Bridging the Best Worlds between Top-k Query and Skyline Query. In *VLDB Journal*.

[9] M. Xie, R. C.-W. Wong, and A. Lall. 2019. Strongly Truthful Interactive Regret Minimization. In *SIGMOD*.

[10] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. 2018. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *SIGMOD*. ACM.

[11] J. Zheng and C. Chen. 2020. Sorting-based interactive regret minimization. In *APWeb-WAIM*.