FindYourFavorite: An Interactive System for Finding the User's Favorite Tuple in the Database

Min Xie, Tianwen Chen, Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
mxieaa@cse.ust.hk, tchenaj@connect.ust.hk, raywong@cse.ust.hk

ABSTRACT

When faced with a database containing millions of tuples, an end user might be only interested in finding his/her favorite tuple in the database. In this paper, we study how to help an end user to find such a favorite tuple with a few *user interactions*. In each interaction, a user is presented with a small number of tuples (which can be artificial tuples outside the database or true tuples inside the database) and s/he is asked to indicate the tuple s/he favors the most among them.

Different from the previous work which displays artificial tuples to users during the interaction and requires heavy user interactions, we achieve a stronger result. Specifically, we use a concept, called the *utility hyperplane*, to model the user preference and an effective pruning strategy to locate the favorite tuple for a user in the whole database. Based on these techniques, we developed an interactive system, called FindYourFavorite, and demonstrate that the system could identify the favorite tuple for a user with a *few* user interactions by always displaying *true* tuples in the database.

ACM Reference Format:
Min Xie, Tianwen Chen, Raymond Chi-Wing Wong. 2019. FindYourFavorite: An Interactive System for Finding the User's Favorite Tuple in the Database. In *2019 International Conference on Management of Data, June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320215

1 INTRODUCTION

In order to assist the user in finding the tuple s/he is interested in, a database system provides some operators to fit the user's need. Such operators can be applied in various domains, including house buying, car purchase and job search. For example, in a car database where each car tuple is described by some attributes, Alice wants to find an inexpensive car with a high horse power, which is as new as possible (i.e., price, horse power and car age are some criteria that Alice would consider when she buys a car). In the literature [1, 2], Alice's preference could be 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. Unfortunately, it is hard for most users to specify their utility functions and even the users themselves might not know their utility functions explicitly.

In this paper, we study how *user interactions* would help in finding the *favorite* tuple for a user in a large database. Intuitively, instead of asking the user for the exact utility function directly (which is difficult for a user to answer), we ask the user to provide "hints" on what his/her utility function might look like (which is easier to provide). Based on the user feedback, we implicitly learn the utility function and determine the tuple that s/he is interested in. In particular, we consider the following type of interaction [1], which is simple and naturally appears in our daily life: *when presented with a short list of tuples, the user is asked to tell his/her favorite tuple (i.e., the maximum utility tuple) among them.*

To motivate the problem, we consider a scenario where an interactive system wants to help Alice to find her favorite car in the market by interactively learning Alice's preference and making recommendations. There can be many candidate cars in the market (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 new car than buying an old one). This trade-off is often individualistic and might not be known by the system in a complete way. To recommend cars effectively, it can show Alice around a short list of cars and Alice can indicate the car she favors the most. 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 this information, the system can filter out those cars definitely uninteresting to Alice. For those cars where Alice's preference is still unknown, another short list of recommendation could be made. By this interactive procedure, Alice can be provided with more and more accurate recommendations and finally, her

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3320215

favorite car could be identified. In this paper, we automate this interactive process, which has many applications such as personalized recommendation and self-guided shopping.

Unfortunately, the existing method in [1] cannot address this problem well, and it has the following major disadvantages: (1) they create artificial tuples (i.e., fake tuples not inside the database) and present those tuples to the user during the interaction, which, however, is not desirable. For example, Alice might be shown a fake car during the interaction and she is attracted by this car. Alice might be encouraged to spend more time on interacting with the system, hoping to obtain an even better car at the end. However, if Alice finally finds that the car she favors is fake, Alice can be disappointed and think that the system is a fraud; (2) they require a large amount of user interactions in order to identify the desired favorite tuple for a user, which is not practical.

To address these limitations, we develop an interactive system called FindYourFavorite, which is powered by our techniques proposed in [5], and it has the following attractive features. Firstly, it provides user-friendly interfaces for users (1) to specify their search constraints, (2) to interact with the system and (3) to visualize the database state (e.g., what are the remaining candidate favorite tuples in the database). Secondly, FindYourFavorite has the desirable guarantee that it always displays true tuples during the interaction. Thirdly, FindYourFavorite can identify the user's favorite tuple with a small amount of interactions and thus, it is more practical and useful. In this paper, we demonstrate FindYourFavorite on a used car database and deploy it to help users to find their favorite cars in this database. To summarize, our major contributions are summarized below:

- We develop an interactive system, FindYourFavorite, for finding the user's favorite tuple in a large database.
- FindYourFavorite provides several interactive interfaces to users and supports various functionalities.
- We deploy FindYourFavorite on a used car database for an easy-to-follow system demonstration.

In the following, we introduce the FindYourFavorite system architecture in Section 2 and demonstrate the system in Section 3. Finally, Section 4 concludes this paper.

2 SYSTEM ARCHITECTURE

The input 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 preferable to all users. If a smaller value is preferable (e.g., price), we can modify the dimension by subtracting each value from the maximum value so that it satisfies this assumption.

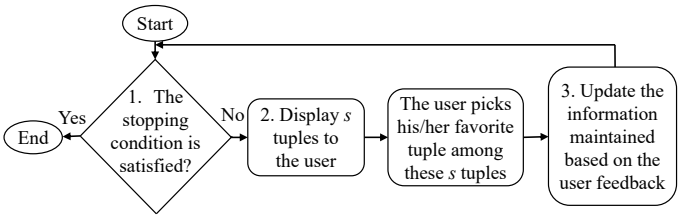


Figure 1: The Interactive System

Same as [1, 5], the user preference can be represented by an unknown *linear utility function*, denoted by f, which is a mapping $f: \mathbb{R}^d_+ \rightarrow \mathbb{R}_+$. A utility 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. In particular, a user is interested in finding the tuple in D which maximizes the utility w.r.t. his/her utility vector and the tuple with the maximum utility is the *favorite* tuple of this user in the whole database. Without loss of generality, we assume that $\sum_{i=1}^{d} u[i] = 1$ in this paper.

2.1 Architecture Overview

The architecture overview of our interactive system, FindYourFavorite, is shown in Figure 1. Specifically, we interact with a user with an *unknown* utility vector for *rounds*. At each round, we ask the user a question by displaying s tuples. After the user picks the tuple s/he favors the most among them, we update the information maintained for learning the user preference until the user's favorite tuple is identified. In particular, we consider the following components:

(1) **(Stopping Condition)** When can we stop interactions?
(2) **(Tuple Selection)** How to select s tuples to display?
(3) **(Information Maintenance)** What types of information should we maintain and how to update the information based on the user feedback?

In the following, we show how we address each of these three components in FindYourFavorite one by one.

2.2 Components Description

Information Maintenance. We maintain two data structures in our solutions: a convex region $\mathcal{R}$ in the utility space which contains the user's true utility vector u and a candidate set $C \subseteq D$ which contains the user's favorite tuple.

Recall that $\sum_{i=1}^{d} u[i] = 1$ and thus, u can be regarded as a (non-negative) point on a hyperplane $H = \{p \in \mathbb{R}^d | \sum_{i=1}^{d} p[i] = 1\}$. We define the *candidate utility range*, denoted by $\mathcal{R}$, to be the convex region on H which contains the user's true utility vector u. For example, in Figure 2 where $d = 3$, $\mathcal{R}$ is the triangular region $\{u \in \mathbb{R}^3_+ | \, u[1] + u[2] + u[3] = 1\}$ before the user provides any information on his/her utility vector u. Given the candidate utility range $\mathcal{R}$, we define the *candidate set* of favorite tuples, denoted by C, to be a subset

of tuples in D such that for each u in $\mathcal{R}$, the favorite tuple of D w.r.t. u is in C. That is, if $p = \arg\max_{q \in D} u \cdot q$ where u is a vector in $\mathcal{R}$, the tuple p is in C. Intuitively, when the user answers more questions, we learn more about his/her utility vector u. Then, $\mathcal{R}$ and the corresponding C will be smaller.

Specifically, if 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 hyperplane $h_{p,q}$ partitions the space $\mathbb{R}^d$ into two halves. The half space above $h_{p,q}$ is denoted by $h^+_{p,q}$. The following lemma shows how we can update $\mathcal{R}$ to be a smaller region based on $h_{p,q}$. Due to the lack of space, the proofs can be found in [5].

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 $h^+_{p,q} \cap \mathcal{R}$.*

Example 2.2. Consider two tuples p and q in Figure 2. We draw the hyperplane $h_{p,q}$ passing through the origin with its unit normal in the same direction as $p - q$. If the user prefers p to q (i.e., $u \cdot p > u \cdot q$), u is in the half space above $h_{p,q}$. $\mathcal{R}$ is then updated to be $h^+_{p,q} \cap \mathcal{R}$ (left sub-triangle). □

Utility hyperplanes are very useful. Firstly, according to the user feedback, we can construct a number of utility hyperplanes and use them to update $\mathcal{R}$. Secondly, based on those utility hyperplanes, we can develop a strategy to prune non-favorite tuples in C, as shown in the following lemma.

Lemma 2.3. *Given $\mathcal{R}$, a tuple q can be pruned from C if there exists a tuple p in C such that $h^+_{q,p} \cap \mathcal{R} = \emptyset$.*

Lemma 2.3 tells us that we can safely prune a tuple q if there is a p in C such that p is preferable to q w.r.t. *any* u in $\mathcal{R}$. We use this strategy to maintain C. Specifically, when $\mathcal{R}$ is updated, we remove each tuple from C that can be pruned by Lemma 2.3 and thus, the remaining C is the candidate favorite tuple set. This process can be implemented by adapting the skyline algorithms [3] and we omit the details here.

Stopping Condition. If there is only one tuple p in C, we conclude that p is the desired favorite tuple according to the definition of C and stop the interaction immediately.

Tuple Selection. We present two approaches of displaying tuples in the interactive system: *Random* and *Simplex*.

Random. At each round, we randomly select s tuples from C and display them to the user. Let p be the favorite tuple picked by the user. For each of the remaining $s - 1$ tuples, namely q, we construct a utility hyperplane $h_{p,q}$, resulting in $s - 1$ utility hyperplanes in total, which update $\mathcal{R}$ and C.

Simplex. The idea is inspired from the geometric interpretation of the Simplex method for Linear Programming [4].

In geometry, the convex hull of D, denoted by $\mathsf{Conv}(D)$, is the smallest convex set containing D. A tuple p in D is a *vertex* of $\mathsf{Conv}(D)$ if $p \notin \mathsf{Conv}(D/\{p\})$. We maintain the vertex $p \in C$ in $\mathsf{Conv}(D)$ with the highest utility displayed so far and interactively check if there is a neighboring vertex of p in $\mathsf{Conv}(D)$ with a higher utility than p by displaying p and at most $s - 1$ neighboring vertices to the user at each round. Each non-favorite tuple corresponds to a new utility hyperplane which will then update $\mathcal{R}$ and C. In this approach, the tuple selection is directed by neighboring vertices in $\mathsf{Conv}(D)$ since they can *strictly* improve the utility of the vertex we maintain, as shown in the lemma below.

Lemma 2.4. *Given a utility vector u and a vertex $p \in C$ of $\mathsf{Conv}(D)$, either p is the favorite tuple w.r.t. u or, there is a neighboring vertex of p, whose utility is larger than that of p.*

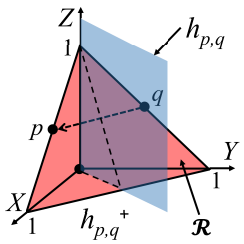
3 SYSTEM DEMONSTRATION

We develop an interactive system called FindYourFavorite based on the proposed techniques. In this section, we demonstrate it on a used car database[1], but our system could also be applied on many other datasets [1, 5] which were not shown due to the lack of space. Recall that Alice wants an inexpensive used car with a high horse power, which is as new as possible. We show how FindYourFavorite helps Alice to identify such a favorite car by asking her a few questions. The three main interfaces of FindYourFavorite are shown in Figure 3 and Alice can use them to (1) specify the search constraints, (2) interact with the system and (3) to visualize the database state. We also shot a video[2] to demonstrate FindYourFavorite under these scenarios. The interested readers could also find our demonstration system online[3].

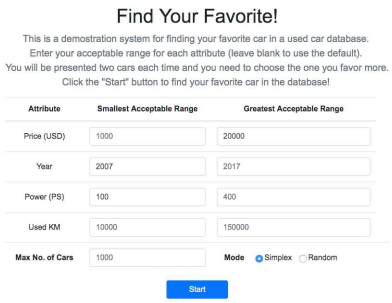
3.1 Constraint Specification

Figure 3(a) shows the constraint specification interface. In this used car database, each car is described by 4 attributes, namely price (in USD), year of purchase, power (in PS) and used kilometers. In general, a lower price and smaller used kilometers are more preferable while a more recent purchase and a higher power are more preferable. Before starting the interaction, Alice can specify some initial constraints on these attributes (Figure 3(a)). For example, Alice may specify the price from 1,000 USD to 20,000 USD (i.e., inexpensive cars), the power from 100PS to 400PS (i.e., high horse power) and the year of purchase from 2007 to 2017 (i.e., as new as possible). Since Alice does not have any requirement on the used kilometers, it is left blank to use the default values. Alice can also specify the maximum number of cars, denoted by $\max_{car}$, that she wants to retrieve for selection and the tuple selection mode (i.e., *Simplex* or *Random*). Suppose that Alice sticks to the default setting of $\max_{car} = 1,000$ and uses the *Simplex* mode. Then, she can click the "Start" button.

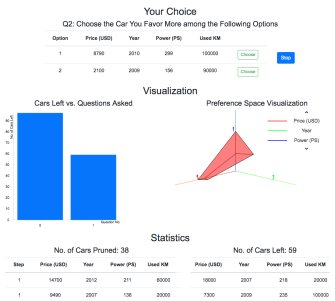
[1] https://www.kaggle.com/orgesleka/used-cars-database
[2] https://youtu.be/FjFbNcQYDFM
[3] https://mxieaa.github.io/FindYourFavorite



(a) Constraint Specification



(b) Interaction



(c) Result

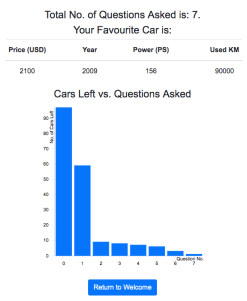


Figure 2: Utility Space

Figure 3: Main Interfaces of FindYourFavorite

After the "Start" button is clicked, FindYourFavorite processes the database and prepares the data for interaction in two steps. Firstly, it retrieves $\max_{car}$ cars from the database which satisfy all the search constraints. Secondly, it preprocesses those cars into the initial candidate set C so that C contains *skyline cars* only [3]. Specifically, a car p is said to *dominate* another car q if p is not worse than q on each attribute and p is better than q on at least one attribute. Then, the utility of p is always higher than that of q (i.e., p is more desirable) regardless of the utility function. Cars which are not dominated by any other cars in the dataset are returned as skyline cars, which form the initial candidate favorite cars of a user (and it conforms to our definition on the candidate set C). For example, after Alice specifies the constraints shown in Figure 3(a), FindYourFavorite constructs the initial candidate set C containing 97 skyline cars for her.

3.2 Interaction and State Visualization

The interaction interface is shown in Figure 3(b). In this stage, FindYourFavorite interacts with the user for rounds and at each round, it asks the user a question by displaying s cars and asking the user to pick the one s/he favors the most. For the purpose of illustration, we set s to 2 to demonstrate the effect of each preference indicated. According to the user feedback, FindYourFavorite updates the data structures (i.e., the candidate utility range $\mathcal{R}$ and the candidate set C) it maintains for finding the user's favorite car and the updated data structures will be visually shown on this interface.

To illustrate, consider the interaction between Alice and FindYourFavorite. Figure 3(b) is the interaction interface *after* Alice is asked one question where in the first question, FindYourFavorite asks Alice for her preference between Car 1 (price=$10,500USD, year=2015, power=110PS, used kilometers=10,000) and Car 2 (price=$8,790USD, year=2010, power=299PS, used kilometers=100,000) and then, Alice clicks on the "Choose" button on Car 2 indicating that she prefers Car 2 to Car 1. After knowing Alice's preference on only a pair of cars, FindYourFavorite filters out 38 cars for her (out of 97 cars in the initial C), which is a 39% reduction on the candidate set size, resulting in only 59 cars in the remaining C (the bottom part in Figure 3(b)). At the same time, FindYourFavorite asks the second question to Alice (i.e., the question shown at the top of Figure 3(b)) for more information about Alice's preference. To better visualize the effect of each preference indicated, we also plot a histogram "Cars Left vs. Questions Asked" and a preference space visualization (i.e., a 3D visualization of $\mathcal{R}$) at the middle of Figure 3(b) so that a user can clearly visualize what FindYourFavorite has done based on his/her feedback.

Finally, when there is only one car in C, it is the user's favorite car and is returned to the user (statistics is also shown). For example, in Figure 3(c), FindYourFavorite identifies the favorite car for Alice by asking 7 questions (i.e., asking her to examine 14 cars) among 1000 candidates in the database. During the interaction, a user can also indicate that s/he wants to stop immediately by clicking the "Stop" button in Figure 3(b) (e.g., when they satisfy with the current C). In this case, the current C is returned to the user.

4 CONCLUSION

In this paper, we show how to find the user's favorite tuple in a large database with the help of user interactions. Based on this idea, we develop an easy-to-use interactive system, FindYourFavorite, and apply it on a used car database, demonstrating our usefulness and effectiveness.

ACKNOWLEDGMENTS

The research is supported by HKRGC GRF 14205117.

REFERENCES

[1] D. Nanongkai, A. Lall, A.D. Sarma, and K. Makino. Interactive regret minimization. In *SIGMOD*, 2012.
[2] D. Nanongkai, A.D. Sarma, A. Lall, R.J. Lipton, and J. Xu. Regret-minimizing representative databases. In *VLDB*, 2010.
[3] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. In *TODS*, 2005.
[4] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems,. In *STACS*, 1992.
[5] M. Xie, R. C.-W. Wong, and A. Lall. Strongly truthful interactive regret minimization. In *SIGMOD*, 2019.