A Confidence-Aware Top-k Query Processing Toolkit on Crowdsourcing

Yan Li[#1] Ngai Meng Kou[#2] Hao Wang[†3*] Leong Hou U[#4*] Zhiguo Gong[#5]

[#]Department of Computer and Information Science, University of Macau, Macau SAR
[1]yb57411@umac.mo [2]yb27406@umac.mo [4]ryanlhu@umac.mo [5]fstzgg@umac.mo

[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[3]wanghao@nju.edu.cn

ABSTRACT

Ranking techniques have been widely used in ubiquitous applications like recommendation, information retrieval, etc. For ranking computation hostile but human friendly items, crowdsourcing is considered as an emerging technique to process the ranking by human power. However, there is a lack of an easy-to-use toolkit for answering crowdsourced top-k query with minimal effort.

In this work, we demonstrate an interactive programming toolkit that is a unified solution for answering the crowdsourced top-k queries. The toolkit employs a new confidence-aware crowdsourced top-k algorithm, SPR. The whole progress of the algorithm is monitored and visualized to end users in a timely manner. Besides the visualized result and the statistics, the system also reports the estimation of the monetary cost and the breakdown of each phase. Based on the estimation, end users can strike a balance between the budget and the quality through the interface of this toolkit.

1. INTRODUCTION

Online crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) and CrowdFlower[1], become popular nowadays, where these platforms can help to distribute simple microtasks [3, 6] (e.g., image classification and labeling) to workers at reasonable costs. For instance, a requester might pay as low as \$0.01 to collect a binary judgment answer. More importantly, these platforms have implemented some well-proven mechanisms to secure the answer quality, e.g., qualification test and gold standard questions.

Some complex tasks, such as joining between two datasets or ranking top-k items, are not natively supported in these crowdsourcing platforms. Therefore, some advanced crowdsourcing systems, e.g., Qurk [9, 10, 11] and CrowdDB [4, 5], were developed to address the need of these tasks. For instance, [10] studied how to support human-powered sorts and joins on the crowdsourcing platforms using an easy-to-use SQL like language.

To the best of our knowledge, most existing work [4, 5, 9, 10, 11] apply a majority vote mechanism for unifying the answers from the crowd workers. However, these work overlook a fact that the cost to unify the answers (i.e., concluding a comparison based on the judgments from the crowd) should be proportional to the difficulty. For instance, picking the younger photo between a child and an adult should need fewer judgments than pick between two adolescents. Based on this observation, our prior work [7] studied a crowdsourced confidence-aware query processing which takes the judgment difficulty (estimated by the confidence level in statistics) into the query processing. As shown in [7], the confidence-aware solution not only secures the query answering quality but also minimizes the crowdsourcing cost.

Our prior study [7] focused on the crowdsourced top-k query [1, 2, 8, 12], that has been considered as an emerging technique for ranking computation hostile but human friendly items, e.g., ranking the 10 most beautiful college campuses in the world. While the top-k query is a crucial requirement in ubiquitous applications, we notice that end users may require some effort to understand the crowdsourcing environment and to learn advanced algorithms before they can really process any top-k queries on the crowdsourcing platforms. For ease of use, we extend our solution [7] to be an easy-to-use programming toolkit so that end users are able to process crowdsourced top-k queries with minimal effort. This toolkit employs a novel confidence-aware algorithm Select-Partition-Rank (SPR) [7] running on a local server or a popular crowdsourcing platform CrowdFlower supporting global crowd channels. We implement the toolkit as a programming library due to the considerations of both user privacy and developer flexibility. In short, users are not willing to share their account information (e.g., the private key in CrowdFlower). Developers, on the other hand, can easily integrate a library into their own systems.

The contributions of this demonstration are threefold: (1) we provide a convenient toolkit to answer the crowdsourced top-k query on a local server or through an online platform; (2) an interactive user interface is designed for fine-tuning the result quality and the monetary budget during the execution; and (3) the intermediate information and statistics are visualized by concise figures. We organize our paper as follows. We give the overview of our toolkit in Sec. 2. Then we introduce our confidence-aware top-k algorithm and our quality control techniques in Sec. 3 and Sec. 4, respectively. The demonstration scenarios are presented in Sec. 5.

*Corresponding authors.

[1]www.mturk.com; www.crowdflower.com

Proceedings of the VLDB Endowment, Vol. 10, No. 12

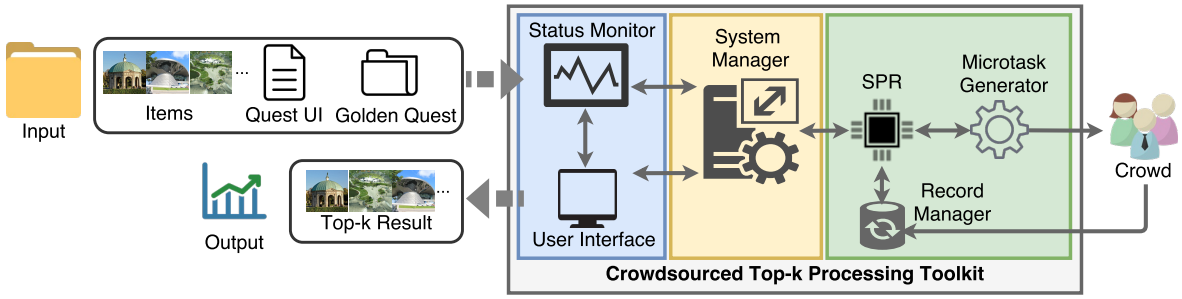


Figure 1: Crowdsourced top-k processing toolkit architecture

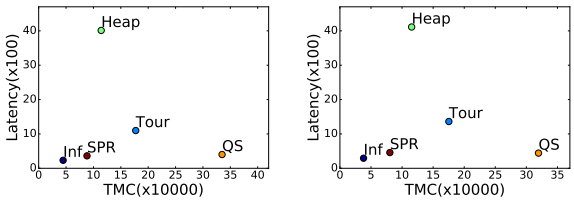
2. OVERVIEW

Figure 1 is an overview of the toolkit architecture. As the execution is a dynamic procedure, it is important to provide functions for monitoring the running statistics, estimating the expected monetary cost, interposing the system at any stage and hiding the complex crowdsourcing platform interface from the end users. It is also critical to provide a strategy to help the users to limit the monetary cost within the budget. Note that, since the monetary cost is proportional to the result quality, it is always possible to find an appropriate confidence level such that the monetary cost is less than the budget limitation.

There are 3 major components in the toolkit. The *user interface* and the *status monitor* in blue color are the functions for interaction and visualization. Users can access the execution status, adjust the budget limitation and update the algorithm parameters in real-time through this component. The *system manager* in yellow color is the component for balancing the monetary cost and result quality. It reads the statistics from the top-k processing algorithm and estimates the expected monetary cost. If the estimated cost exceeds the budget limitation, it will pause the algorithm and wait for the user's decision (semi-automatic mode) or directly re-balance the cost and quality by reducing the objective confidence level (automatic mode). The component in green color is the back-end processing logic. The execution strategy is based on SPR algorithm. *Record manager* stores the results of all the finished microtasks (i.e., comparisons) to enable fault-tolerant processing. *Microtask generator* and *platform interface* transform an internal comparison to worker readable microtasks and communicate with crowd.

3. SELECT-PARTITION-RANK

Select-Partition-Rank (SPR) [7] is a confidence-aware algorithm for the crowdsourced top-k query. The experiment results in Figure 2 show that SPR is one of the most monetary and time efficient solutions to find the top-k items with quality guarantee. Therefore we implement SPR in the toolkit as the query processing algorithm. SPR decomposes the top-k query into a set of pairwise preference comparisons and distributes the comparisons as microtasks to the crowd. The worker who accepts a microtask needs to decide her preferred item and indicate her preference level (i.e., how much she prefers the item). Since some comparisons can be answered easily (i.e., one item is much better than the other) while some comparisons may be tough (i.e., two items are similar), SPR dynamically decides the workload of each comparison to make sure that every comparison result has the same confidence. So the total monetary cost of SPR is proportional to the total workload of all the comparisons. SPR has 3 phases: *selecting* phase, *partitioning* phase and *ranking* phase. We shortly introduce the basic idea of each phase in the following.



(a) IMDb dataset (b) Book dataset

Figure 2: Performance summary of SPR with regard to the total monetary cost (TMC) and the query latency. Inf is the optimal yet infeasible solution. Heap, Tour and QS are the baseline solutions.

Select. Given the item set $\mathcal{O}$ with size N, let $\{o_1^*, o_2^*, \cdots, o_k^*\}$ be the top-k items where $o_i^* \in \mathcal{O}$ and $o_i^* \succ o_j^*$ (o_i^* is superior to o_j^*) for any $i < j$. SPR wants to find a reference item in $\{o_k^*, o_{k+1}^*, \cdots, o_{ck}^*\}$ by sampling where $c > 1$ and $ck \ll N$. This reference can be found by taking the *median* item from a set of m candidates where each candidate is the *max* item of x random samples from $\mathcal{O}$ [7].

Partition. We compare each item $o \in \mathcal{O} \setminus r$ with the reference r. Since $r \in \{o_k^*, o_{k+1}^*, \cdots, o_{ck}^*\}$, we can safely filter the items which lose the comparison with r. There will be only a small set of items remained.

Rank. In the ranking phase, the remaining items are ranked by a sorting method with near linear performance. The top-k items can then be returned.

4. ESTIMATION AND SELF-ADAPTATION

In this section, we discuss how the end users can strike a balance between the monetary budget and the top-k answering quality through the *system manager* of this toolkit.

Almost every crowdsourcing algorithm has its preferred parameter setting which is well-tuned for the testing applications, but the default settings are not enough for an open system due to various application domains. Tuning the parameters is painful and sometimes impossible for the end users. With inappropriate settings, the total monetary cost and the result quality will not be guaranteed. In this toolkit, we implement two features, budget estimation and quality

self-adaptation, to avoid parameter tuning and make budget controllable under certain confidence level.

Budget estimation. SPR is a confidence-aware algorithm that its result quality is subject to the objective confidence level. We only need to estimate (i.e., forecast) the total monetary cost according to the confidence level and detect if there is a risk of exceeding budget. The total monetary cost (TMC) in SPR can be measured by

$$\text{TMC} = \sum_{\text{COMP}(o_i, o_j) \in \mathcal{C}} w_{i,j},$$

where $\mathcal{C}$ is the set of pairwise comparisons $\text{COMP}(o_i, o_j)$, and $w_{i,j}$ is the workload (number of judgments) for comparison.

The estimation of Estimated Cost (EC) consists of 3 components, estimating the cost of the selecting phase, the partitioning phase and the ranking phase. The Estimated Cost (EC) of each phase can be divided into 2 parts, the actual current cost and the estimated future cost. Let AC_s, AC_p and AC_r be the actual current cost of selecting phase, partitioning phase and ranking phase. And let EC_s, EC_p and EC_r be the estimation of future cost of the corresponding phases. Then we have

$$\text{EC} = \text{AC}_s + \text{EC}_s + \text{AC}_p + \text{EC}_p + \text{AC}_r + \text{EC}_r.$$

For every phase, the actual cost is known and the future cost can be estimated by the product of the expected workload of a comparison and the number of expected future comparisons of that phase. Let $\bar{w}$ be the average workload of the past comparisons and we use $\bar{w}$ to approximate the expected workload of a comparison for simplicity. The expected number of comparisons in each phase can be bounded according to the execution procedure. The number of future comparisons is the expected number of comparisons excluding the ones already finished. Next, we summarize the expected number of comparisons phase by phase.

In the selecting phase, we select the *median* item from m independent candidates as a reference r where each candidate is the *max* item from x item samples. x and m are decided based on a limitation that the expected number of comparison is bounded by N, which is also the expected number of comparisons [7] in the selecting phase. In the partitioning phase, we compare r with every other items which indicates the expected number of comparisons is $N - 1$. In the ranking phase, with the reference-based sorting, the expected number of comparisons is approximated by $2N$. The total monetary cost can then be estimated.

Self-adaptation. Due to inappropriate settings, the estimation of the total monetary cost may exceed the budget. In this case, the toolkit provides an interactive interface to balance the quality and the cost. The user has to adjust the budget and the confidence level till the estimation of the total monetary cost is less than the budget[2]. There are two methods, one to increase the budget and the other to decrease the confidence level. Budget increase is done manually by the user while the confidence level can be tuned automatically based on a self-adapting mechanism.

Since the expected number of comparisons is bounded, we then analyze the relationship between the confidence level $1 - \alpha$ and the expected workload of a comparison. For each finished comparison (of w samples), let $\bar{\mu}_w$ be the sample mean and S_w be the sample standard deviation. Suppose without loss of generality $\bar{\mu}_w > 0$, the expected workload w' under a new confidence level $1 - \alpha$ can be derived [7] as follows:

$$w' = \left(\frac{S_w \cdot t_{\frac{\alpha}{2}, w-1}}{\bar{\mu}_w} \right)^2,$$

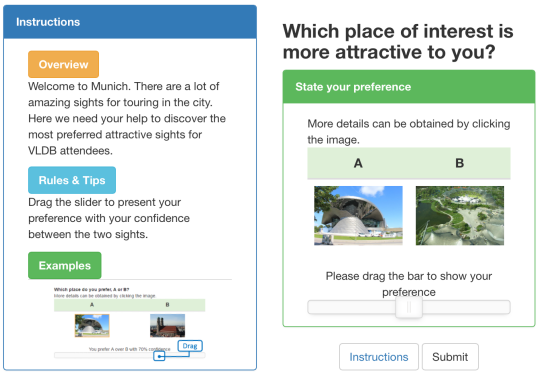
where $t_{\frac{\alpha}{2}, w-1}$ indicates the right-tail probability of size $\frac{\alpha}{2}$ of the t-distribution with $w - 1$ degrees of freedom. Thus, the confidence level can be adjusted to the maximum value that strikes a balance between the cost and the quality.

[2]In practice, the budget should be larger than $1.1 \cdot \text{EC}$ after balance to avoid repetitious adjustment.

5. DEMONSTRATION

In this demonstration, we invite all attendees to involve in a ranking game, *"what are the most attractive sights for the VLDB 2017 attendees?"* on a local server and show the running procedure of the toolkit including all discussed features. Our audiences can participate the game either on their laptop or smart phone. We start the demonstration by showing the ranking game instruction. In the pairwise judgment page, each audience states her preference for the sights by dragging a slider bar. If an audience wants to know more about the sight, she can check the Wikipedia page by clicking on the image. Meanwhile, our audiences can investigate the entire query processing, from the input data formalization to the final result collection, via the status monitor interface.

5.1 Demonstration Setup



(a) Instruction (b) Judgment

Figure 3: Quest UI

For a top-k query, a user needs to prepare a quest interface and an item list. The quest interface is an HTML page for pairwise judgments. Figure 3 shows the quest interface in the ranking game, where two items can be judged by dragging the slider bar. To support different kinds of items, each item in the item list is represented by a tuple $\langle \text{ItemID}, \text{URL} \rangle$ where ItemID is a unique index for the item and URL is the entity (e.g., an image, a video, a word, etc.). For the sake of demonstration, we select 20 photos of famous sights in Munich as the item list and preload into our local server. We also assume that our attendees will answer the judgments to the best of their knowledge (i.e., no golden quest is needed in the ranking game).

5.2 Status Monitor

During the execution, audience can monitor the current running query via the status monitor. We assume each answer collected from the VLDB 2017 attendees costs $0.01.

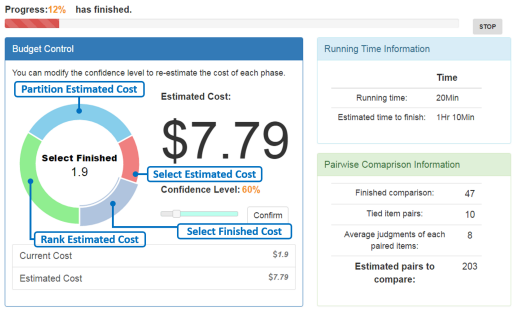


Figure 4: Status monitor - upper windows

Progress monitor. In the upper windows (Figure 4), the overall progress of the task can be view on the top of the interface. On its right, there is a "stop button" to terminate a query immediately. The right side shows the statistics of the query execution, such as running time, estimated running time, number of comparisons, number of ties, average workload and estimated number of pairs to compare.

Budget Control. In the budget control window, the estimation of the total monetary cost is highlighted while its breakdown is illustrated by a doughnut graph. The current cost and the estimated future cost of every phase of SPR is shown as a piece of doughnut. Users can investigate the changes in real time. The slider bar below the estimated total monetary cost is used to tune confidence level.

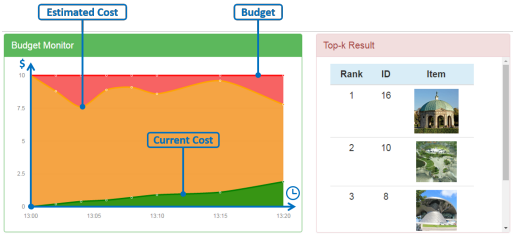
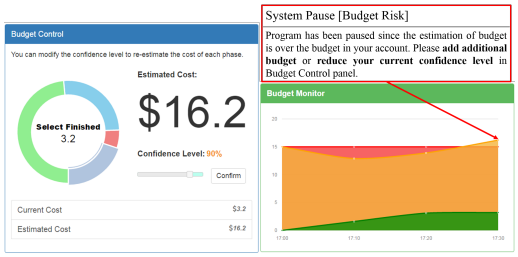


Figure 5: Status monitor - lower windows

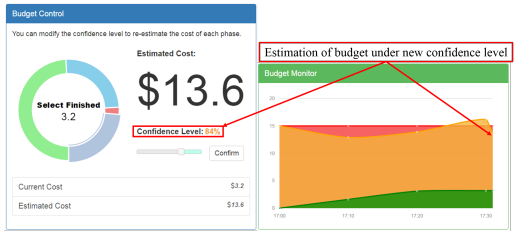
Budget monitor. Figure 5 is the lower windows of the status monitor. In the left hand side, there is a line chart of the current monetary cost (green line), the estimated future monetary cost (orange line) and the total budget (red line) over time. The current monetary cost and the estimated future monetary cost are updated when a new microtask result is collected.

Result monitor. On the right hand side of Figure 5 is the result monitor. It shows a list of the top-k items based on current collected result. The list will be updated over time during the execution.

Interaction example. Figure 6 shows an example that users control the monetary cost under the budget via an interactive process. At some time point, even the current cost ($3.2) is under the budget ($15), the estimated cost ($16.2) is over the budget limitation. The program pauses the execution immediately and throws out a budget risk as



(a) Budget risk



(b) Control budget with confidence level

Figure 6: An example of budget control

shown in Figure 6(a). In order to continue the process, the user could add an additional budget or reduce the confidence level. Figure 6(b) shows the result of taking the latter option. The confidence level is automatically reduced from 90% to 84%. The estimation of budget is then decreased from $16.2 to $13.6 correspondingly.

Acknowledgement. This work was supported by MYRG-2016-00182-FST and MYRG2014-00106-FST from UMAC RC, 61502548, 61432008, 61503178 from NSFC, BK20150587 from Jiangsu Province, FDCT/007/2016/AFJ from FDCT.

6. REFERENCES

[1] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[2] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4):35:1–35:39, 2014.

[3] A. Doan, M. J. Franklin, D. Kossmann, and T. Kraska. Crowdsourcing applications and platforms: A data management perspective. *PVLDB*, 4(12):1508–1509, 2011.

[4] A. Feng, M. J. Franklin, D. Kossmann, T. Kraska, S. Madden, S. Ramesh, A. Wang, and R. Xin. Crowddb: Query processing with the VLDB crowd. *PVLDB*, 4(12):1387–1390, 2011.

[5] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[6] B. Frei. Paid crowdsourcing. *Current State & Progress toward Mainstream Business Use, Smartsheet. com Report, Smartsheet. com*, 9, 2009.

[7] N. M. Kou, Y. Li, H. Wang, L. H. U, and Z. Gong. Crowdsourced top-k queries by confidence-aware pairwise judgments. In *SIGMOD*, pages 1415–1430, 2017.

[8] J. Lee, D. Lee, and S. Hwang. Crowdk: Answering top-k queries with crowdsourcing. *Inf. Sci.*, 399:98–120, 2017.

[9] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for human operators. In *SIGMOD*, pages 1315–1318, 2011.

[10] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[11] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[12] V. Polychronopoulos, L. de Alfaro, J. Davis, H. Garcia-Molina, and N. Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.