ISSN 1330-3651(Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20190124092015
Original scientific paper

An Efficient Top-k Query Scheme Based on Multilayer Grouping

Zongmin CUI, Yu GAO, Caixue ZHOU, Guangyong GAO, Zhuolin MEI, Zongda WU

Abstract: The top-k query is to find the k data that has the highest scores from a candidate dataset. Sorting is a common method to find out top-k results. However, most of existing methods are not efficient enough. To remove this issue, we propose an efficient top-k query scheme based on multilayer grouping. First, we find the reference item by computing the average score of the candidate dataset. Second, we group the candidate dataset into three datasets: winner set, middle set and loser set based on the reference item. Third, we further group the winner set to the second-layer three datasets according to k value. And so on, until the data number of winner set is close to k value. Meanwhile, if k value is larger than the data number of winner set, we directly return the winner set to the user as a part of top-k results almost without sorting. In this case, we also return the top results with the highest scores from the middle set almost without sorting. Based on above innovations, we almost minimize the sorting. Experimental results show that our scheme significantly outperforms the current classical method on the performance of memory consumption and top-k query.

Keywords: almost minimizing sorting; multilayer grouping; top-k query

1 INTRODUCTION

Top-k query [1] returns the best k results to the user from a large amount of data [2].Top-k query has a wide range of applications [3], such as e-commerce [4], car networking [5] and other fields. Recently, Google, Twitter, Facebook also need to use Top-k query to collect the top data.

In the current top-k query methods, SPR [6] has three shortcomings: high sorting costs, insufficient grouping and high memory consumption. The sorting cost of tree selection [7] is also high. The memory consumption of heap sorting [8] is also high. To remove the above issues, we propose a Top-k query scheme based on Multilayer Grouping. We name our scheme TMG. Experimental results show that we enhance the top-k query efficiency and reduce the memory consumption. However, we consume more index building time than SPR.

Our contributions are illustrated as follows.

(1) We propose a method almost without sorting. We group the candidate dataset into three datasets: winner set, middle set and loser set. In the worst case, only one set is needed to find the top results almost without sorting. Thus, we decrease the sorting cost.

(2) We propose a multilayer grouping frame. When the data number of candidate dataset is much larger than k value, we further group the winner set to the second-layer three datasets according to k value. And so on, until the data number of winner set is close to k value. Therefore, we decrease the data number of the candidate dataset to decrease the sorting cost.

(3) We propose a method that removes the redundant data from regrouping. When the data number of winner set is less than k value, we do not regroup the middle set and loser set. Therefore, we remove the redundant data from regrouping to decrease the memory consumption.

The rest of this paper is organized as follows. In Sections 2, we review the related works. In Section 3, we show our core algorithms to almost minimize the sorting cost. In Section 4, we compare the TMG with the existing method by experiments. In Sections 5, we conclude this paper.

2 RELATED WORKS

2.1 Grouping Sort

The most classic method of grouping sort is proposed by Kou etc. [6]. Their method is called SPR. After pair-preference judgments, each candidate item gets a preference value package. The preference value package of each candidate is estimated by Gauss distribution. The confidence interval of the estimated value is used to judge the reliability of the estimated value. First, SPR computes the average score of the maximum value in random sampling sets to find the reference item r. Second, SPR computes the best range c based on r. They give the definition and proof of c. Third, they group the candidate dataset into three datasets: winner set, middle set and loser set. If a data's score is bigger than c's maximum value, the data is put into the winner set. If a data's score is in c, the data is put into the middle set. If a data's score is smaller than c's minimum value, the data is put into the loser set. Fourth, they sort the winner set. If the data number of winner set is larger than k value, they return the top-k results to the user. Obviously, the top-k results have the highest k scores. If the data number of winner set is smaller than k value, they connect middle set and loser set as a new candidate dataset. Finally, they regroup the new candidate dataset to get the new three datasets, and so on, until they find the k highest-score results. For clearer description, we take the following example to analyze their shortcomings.

Example 1. The winner set has 10,000 data, the middle set has 12,000 data, and the loser set has 30,000 data.

(1) High sorting costs. No matter what k value is, SPR needs to sort the winner set. For example, if k = 10005, SPR sorts winner set's 10,000 data. It is clear that the 10,000 data do not have to be sorted at all. In this case, we directly return the winner set to the user almost without any sorting. For another example, k = 5, SPR sorts the 10,000 data, and then returns the top-5 highest-score data to the user. In this case, we directly return the top-5 results almost without sorting. Based on above innovations, our scheme TMG decreases the sorting cost.

(2) Insufficient grouping. For example, if k = 5, finding top-5 highest score data by sorting 10,000 data is

not easy. In other words, SPR is not efficient enough for top-k query. To enhance the efficiency, we propose a multilayer grouping frame. In Example 1, we further group the winner set to the second-layer three datasets. Our second-layer winner set may have only 3000 data. The third-layer winner set may have only 1000 data, and so on. The last-layer winner set may have only 6 data. Obviously, finding top-5 highest score data from 6 data is quite easy. Therefore, our scheme TMG enhances the top-k query efficiency and decreases the memory consumption.

(3) Redundant data for regrouping. For example, if $k = 10005 > 10000$, SPR connects middle set and loser set as a new candidate dataset. Then SPR regroups the 42000 data to get the new three datasets. Obviously, loser set's 30,000 data is unlikely to have the highest scores. In other words, the 30,000 data is redundant for regrouping. However, we only find the top-5 highest-score data from the middle set rather than regrouping them. Thus, our scheme TMG removes the redundant data and improves the grouping efficiency.

2.2 Tree Selection Sort

Tree selection sort methods [7, 9, 10] are widely used in top-k query processing. First, these methods randomly divide all data into $N/2$ pairs. After a pair comparison, a better one rises to the up-layer of the tree until the best one reaches the root. The second best one can be determined by constructing another tree selection sort. And so on, until top-k results are all found. Tree selection sort methods are more demanding for initial sorting. Therefore, these methods have a high sorting cost. However, we almost minimize the sorting cost. Thus, our scheme TMG has a fewer sorting cost than these methods.

2.3 Heap Sort

Heap sort methods [8, 11, 12] initialize a minimum heap with k immediate data. Then, they test each of the remaining data closest to the top of the heap in order. Once they find that there is a better data than the worst candidate, they replace the worst top-k candidate by the better data. That is, the better data becomes the new top-k candidate's. In addition, so on, until the last top-k result is found. When the initial sort does not reach the best heap row, the memory consumption of these methods is high. We almost minimize the sorting cost. Therefore, our scheme TMG has a smaller memory consumption than heap sort methods.

3 TOP-K QUERY SCHEME

We take Fig. 1 as a running example to illustrate our core idea. Fig. 1 is an example of candidate dataset D. In general, our data in a candidate dataset D is from a set of crowd sourced pairwise judgments [13-15].

In Fig. 1, D has 18 candidate data: $d_1, d_2, \ldots, d_{18}$. We use $d_i.s$ to denote d_i's score. For example, $d_2.s = 8.5$. Our challenge is to find the top-k highest-score data as efficiently as possible.

The core idea of TMG is to multilayer group the candidate dataset based on k value to almost minimize the top-k query cost. According to k value, we first compute the average score of candidate dataset D as reference item r. Second, we compute best range c based on r according to literature [6]. Third, we group the candidate dataset into three datasets: winner set, middle set and loser set based on best range c. According to k value, we further group the winner set. And so on, until the data number of winner set is close to k value. Finally, the top-k query operation is performed on the last-layer grouping. Therefore, this paper involves the following three algorithms: Grouping, Top-k, and Query.

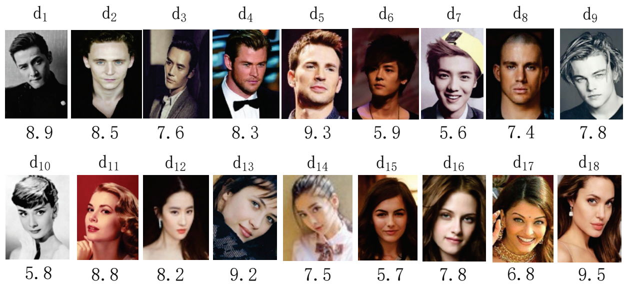


Figure 1 An example of candidate dataset D

3.1 Grouping

Algorithm 1 is a grouping algorithm used to group the candidate dataset into three datasets: winner set, middle set and loser set. The algorithm takes the candidate dataset D as input and takes the winner set (WS), middle set (MS) and loser set (LS) as outputs. First, the algorithm computes the reference item r (Steps 5-8). Second, it computes the best range c (Step 9). Third, it groups D based on c (Steps 10-18).

```
Algorithm 1: Grouping
Input: $D$
Output: WS, MS, LS
1:  WS:= $\emptyset$ //Winner set
2:  MS:= $\emptyset$ //Middle set
3:  LS:= $\emptyset$ //Loser set
4:  D.S:=0 //Sum of all scores
5:  For all $d_i \in D$ do
6:      D.S:= D.S+$d_i.s$
7:  End for
8:  r:=Round(D.S/|D|, 1)
9:  Compute best range c by r according to literature [6]
10: For all $d_i \in D$ do
11:     If $d_i.s$>$c$.max then
12:         WS:=WS ∪ $d_i$
13:     Else if $d_i.s \in c$ then
14:         MS:=MS ∪ $d_i$
15:     Else
16:         LS:=LS ∪ $d_i$
17:     End if
18: End for
19: Return(WS, MS, LS)
```

We take Fig. 1 as an example to illustrate Algorithm 1. First, the algorithm computes the average score of the 18 data in Fig. 1. The average score is 7.7. Thus, reference item $r = 7.7$ (Steps 5-8). Second, it computes the best range $c = [7.0, 8.2]$ based on r according to literature [6] (Step 9). In this case, the maximum value of

c is c.max = 8.2. Finally, 18 data is grouped according to c (Steps 10-18). For example, as $d_1.s$ = 8.9 > c.max = 8.2, d_1 is put into winner set WS. $d_{12}.s$= 8.2∈c = [7.0, 8.2], thus d_{12} is put into middle set MS.$d_{17}.s$ =6.8 < c.min = 7.0, thus d_{17} is put into loser set LS. Based in the same way, the final grouping results are shown in Fig. 2.

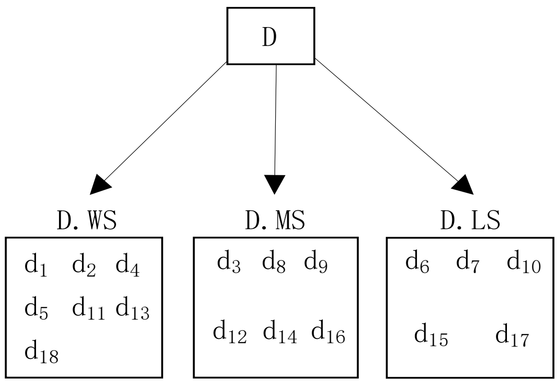

D
D. WS
D. MS
D. LS
d_1 d_2 d_4 d_5 d_{11} d_{13} d_{18}
d_3 d_8 d_9 d_{12} d_{14} d_{16}
d_6 d_7 d_{10} d_{15} d_{17}

Figure 2 An example of first-layer grouping

3.2 Top-k

Algorithm 2 is used to find the highest-score k data from dataset W. The algorithm takes dataset W and k value as inputs and the top-k result set (TOPK) as output.

First, Algorithm 2 assigns the k data that has top serial number to TOPK (Step 4). Meanwhile, it removes the k data from the original dataset W (Step 5). The remaining data in W is then compared to the data in TOPK (Step 9). If remaining data d_j has a bigger score than TOPK data d_m we replace d_m by d_j (Steps 10-11). That is, the better data belongs to the new top-k candidate. In the same way, we can find the top-k results.

```
Algorithm 2: Top-k
Input: $W$,$k$
Output: TOPK
1:  TOPK:= ∅ //Highest-score k results
2:  i:=1
3:  For all $i≤k$ do
4:      TOPK:=TOPK∪ $d_i$
5:      W:=$W/d_i$
6:  End for
7:  If W ≠ ∅ then
8:      For all $d_j ∈ W$ do
9:          If TOPK's minimum value$d_m.s < d_j.s$ then
10:             TOPK:=TOPK/$d_m$
11:             TOPK:=TOPK∪ $d_j$
12:         End if
13:     End for
14: End if
15: Return(TOPK)
```

For example, we want to find the top-3 results from the middle set of Fig. 2. Then, we run Top-k(D.MS, 3). First, we assign the 3 data that has top serial number to TOPK. That is, TOPK = {d_3, d_8,d_9} and W = {d_{12}, d_{14}, d_{16}} (Steps 4-5). As $d_3.s$ = 7.6, $d_8.s$ = 7.4, $d_9.s$ = 7.8, TOPK's minimum value $d_8.s$ = 7.4 < $d_{12}.s$ = 8.2. Thus, TOPK = {d_3, d_{12},d_9}. In the same way, finally, TOPK = {d_{16}, d_{12},d_9}.

3.3 Query

Based on Algorithms 1 and 2, Algorithm 3 shows our core idea. Algorithm 3 takes the candidate dataset D and k value as inputs and queried top-k result set R as output.

First, if the number of data in candidate dataset D is much larger than k value (Step 3), Algorithm 3 needs to group D (Step 4) to almost minimize the data number of candidate dataset P (Step 2). Otherwise, if the number of data in candidate dataset D is close to k value, there is no need to group. Second, if the data number of winner set is larger than k value and at the same time is not much larger, we find the top-k results from the winner set almost without sorting (Steps 5-6). Third, if the data number of winner set is smaller than k value (Step 7), the sum of data number of the winner set and the middle set should be larger than k value. Thus, we directly return the winner set to the user as a part of top-k results almost without sorting (Step 8). Meanwhile, we also find the k-|P.WS| data with the highest scores from the middle set (Step 9). It is worth mentioning that the middle set does not need to be regrouped, because the data number of the middle set is close to k value in this case. Finally, the winner set is regrouped to the second-layer three datasets (Step 9). And so on, until the data number of winner set is close to k value. Based on above innovations, we almost minimize the sorting cost.

```
Algorithm 3: Query
Input: $D$, $k$
Output: R
1:  R:= ∅ //Top-k results
2:  P:=$D$ //Candidate dataset
3:  While |P|>2$k$ do
4:      Grouping(P) //Algorithm 1
5:      If $k$ ≤ |P.WS|≤ 2$k$ then
6:          R:=R∪Top-k(P.WS, k) //Algorithm 2
7:      Else if |P.WS|<$k$ then
8:          R:=R∪P.WS
9:          R:=R∪Top-$k$(P.MS, $k$-|P.WS|)
10:     End if
11:     P:=P.WS
12: End while
13: Return(R)
```

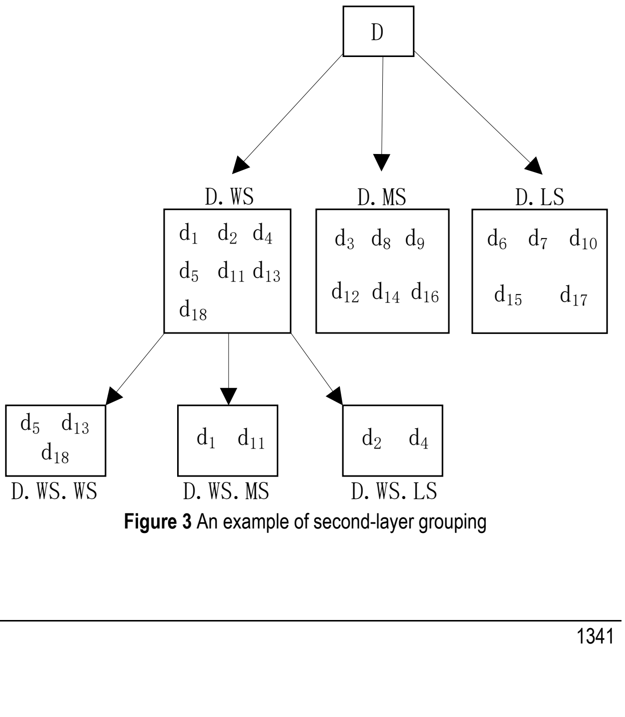

D
D. WS
D. MS
D. LS
d_1 d_2 d_4 d_5 d_{11} d_{13} d_{18}
d_3 d_8 d_9 d_{12} d_{14} d_{16}
d_6 d_7 d_{10} d_{15} d_{17}
d_5 d_{13} d_{18}
d_1 d_{11}
d_2 d_4
D. WS. WS
D. WS. MS
D. WS. LS

Figure 3 An example of second-layer grouping

We take Fig. 1 as an example to illustrate Algorithm 3. That is, we run Query (D, 3), i.e. $k = 3$. First, D has 18 data, i.e. $|D| = 18 > 2k = 6$ (Step 3). Second, we group D shown in Fig. 2 (Step 4). In Fig. 2, |D.WS| = 7 > 6, thus we have the second-layer group winner set D.WS shown in Fig. 3 (Step 11). In Fig. 3, |D.WS.WS| = 3 = k, thus Query(D, 3) = $\{d_5, d_{13}, d_{18}\}$ almost without sorting.

3.4 Analysis

Almost all of query methods are not proposed for only one single query, but for a large number of queries. Therefore, the query index (e.g Fig. 3) should be stored in memory to support real-time query. If k value is large, we need to use a high-layer grouping. For example, $k = 8$ in Fig. 3, there is no need to use the second-layer grouping. We just need to use the first-layer grouping. If k value is small, we need to use a low-layer grouping. For example, $k = 2$ in Fig. 3, we need to use the second-layer grouping. If k value is very small, we even need to regroup the winner set.

SPR ,only single-layer, groups the candidate dataset. Thus the index building complexity of SPR is $O(k \times |D|)$. Our scheme TMG requires multiple-layer grouping it. Thus, TMG's index building complexity is $O(k \times |D| \times \log_2|D|)$. When the index is built, SPR always sorts the winner set during a top-k query. Thus, SPR's stack using complexity is $O(k \times |D|^2)$. TMG almost minimizes the sorting cost, thus our stack using complexity is $O(k \times |D| \times \log_2|D|)$. Memory consumption consists mainly of two parts: index building and stack using. Therefore, SPR's memory consumption complexity is $O(k \times |D| + k \times |D|^2)$. Our scheme TMG's memory consumption complexity is $O(2 \times k \times |D| \times \log_2|D|)$. When the index is built, SPR does not use the multilayer grouping, so SPR needs to query a lot of data. Meanwhile, SPR always sorts the winner set. Thus, SPR's Top-k query complexity is $O(k \times |D|^2)$. We use the multilayer grouping to almost minimize the sorting cost. Thus, TMG's Top-k query complexity is $O(k \times (\log_2|D|)^2)$. Obviously, our scheme has better performance than SPR. The comparison of complexity is shown in Tab. 1.

Table 1 The comparison of complexity

	Index building	Memory consumption	Top-k query
SPR [6]	$O(k \times \lvert D\rvert)$	$O(k \times \lvert D\rvert + k \times \lvert D\rvert^2)$	$O(k \times \lvert D\rvert^2)$
TMG	$O(k \times \lvert D\rvert \times \log_2\lvert D\rvert)$	$O(2 \times k \times \lvert D\rvert \times \log_2\lvert D\rvert)$	$O(k \times (\log_2\lvert D\rvert)^2)$

4 EXPERIMENTS

The SPR method [6] is most relevant to our scheme TMG. Meanwhile, SPR is also a very classical method for our application scenarios. Therefore, to prove the efficiency and practicality of our scheme, this part compares SPR with TMG about the index building, memory consumption and top-k query.

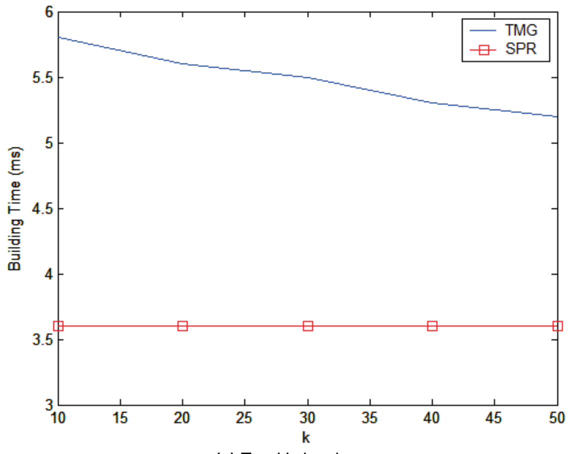
4.1 Experimental Setup

Our experimental candidate dataset D is from a set of crowd sourced pairwise judgments. We store D in a MySQL database. Visual C++ 6.0 is used in all experiments. The experiment uses a computer with a 3.4 GHZ dual-core CPU and 32 GB of memory. Following the general setting of the traditional top-k query system, we assume that the index is stored in memory to support real-time response [16]. When we compare SPR with TMG, they are always in the same dataset state.

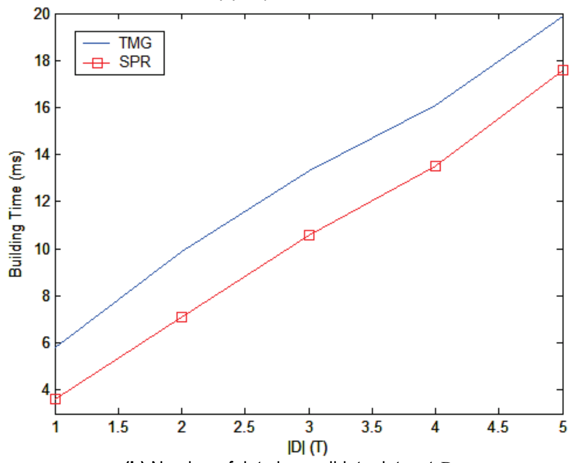
Table 2 The specific parameters

Parameter	Values
k	10, 20, 30, 40, 50
$\lvert D\rvert$	1T, 2T, 3T, 4T, 5T

The k value in the system is from 10 to 50. The Number of data in candidate dataset D is from 1T to 5T (T denotes thousand). The specific parameters are shown in Tab. 2. When we change a parameter, the remaining parameter is the default setting (shown in bold).



(a) Top-k's k value



(b) Number of data in candidate dataset D

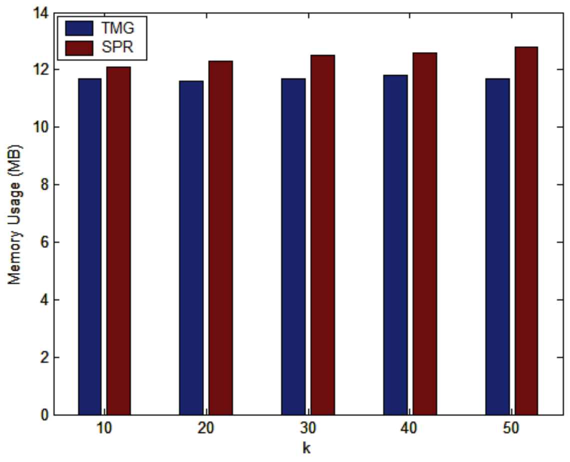
Figure 4 The comparison of index building times

4.2 Index Building

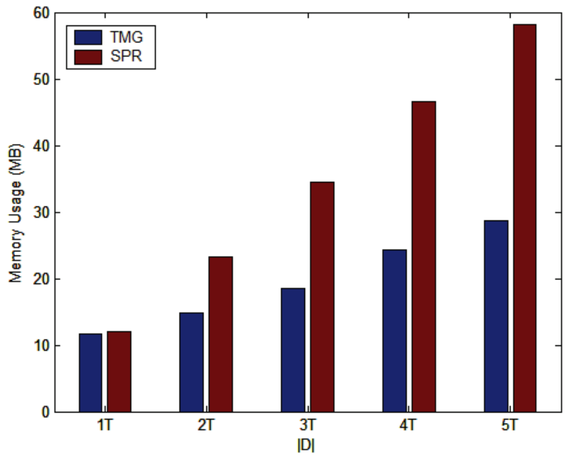
Before beginning a top-k query, we should establish a query index to support the query. The index building process is mainly the data grouping process. For this reason, our first set of experiments tests the building time of these two methods. The test results are shown in Fig. 4, where the Y-axes represent the building time, whose unit is millisecond (ms). The X-axes of Fig. 4(a) and Fig. 4(b) represent k value and D's data number $|D|$ (whose unit is thousand) respectively.

(1) Fig. 4(a). In the process of k value increasing from 10 to 50, the *k* value is much smaller than |*D*| (This is a kind of most common cases in practical applications). In these cases, SPR does not need to regroup middle set and loser set. All queries are performed on the initial grouping. As a result, there is almost no change about the building time of SPR. For our scheme TMG, the number of regrouped layers decreases with the increasing *k* value, which reduces the regrouping number. Therefore, our building time decreases with the increasing *k* value.

(2) Fig. 4(b). In the process of |*D*| increasing from 1T to 5T, our scheme TMG always consumes more building time than SPR. This is because we have a multilayer grouping and SPR only has a single-layer grouping. Meanwhile, in our scheme, the deeper layer results in fewer data used for grouping. Therefore, our building time increases almost linearly. There is not much more building time than SPR.



(a) Top-k's kvalue



(b) Number of data in candidate dataset *D*

Figure 5 The comparison of memory consumption

4.3 Memory Consumption

Memory consumption consists mainly of two parts: index building and stack using. In general, the less memory consumption brings the higher computing efficiency of the system. Therefore, we test the memory consumption of TMG and SPR in the event that the index building is complete. In this case, the experimental results are shown in Fig. 5, where the *Y*-axis represents the memory consumption in MB. The *X*-axis in Fig. 5(a) and Fig. 5(b) represents top-k value and *D*'s number of data in |*D*|.

(1) Fig. 5(a). Obviously $k = 50 \ll 1T$, so when the *k* value increases from 10 to 50, SPR always needs to sort the winner set. Thus, there is little change in stack using and no change in the storage consumption of the index. Consequently, there is little change in memory consumption of SPR. Our scheme TMG groups data according to *k* value. The data number of our regrouped candidate dataset is much fewer than SPR. Thus, the memory consumption of our stack using increases slowly as *k* value increases. That is, our index memory consumption remains almost constant as *k* value increases. Since we almost minimize the sorting cost, we consume less memory than SPR.

(2) Fig. 5(b). When |*D*| increases from 1T to 5T, SPR needs to sort the winner set. Thus, the memory consumption is increasing rapidly as |*D*| increases. Through the multilayer grouping, the data number that we need to sort is determined by *k* value. Thus, the memory consumption of our stack using changes little. Meanwhile, our index memory consumption increases as |*D*| increases. Therefore, our memory consumption growth rate was significantly lower than that of SPR.

4.4 Top-k Query

Query efficiency is actually to test the time consumed by finding Top-k results, which is the most critical performance of the system. The key of top-k query is *k* value. Therefore, only when the user submits the query request with *k* value, can the whole query process begin. When the index has been established and stored in memory, we can quickly support real-time online queries. Therefore, the time consumed by queries in this part only includes the query time.

The experimental results are shown in Fig. 6, where the *Y*-axes represent the query time, whose unit is microsecond (μs). The *X*-axes of Fig. 6(a) and Fig. 6(b) represent *k* value and *D*'s data number |*D*| respectively.

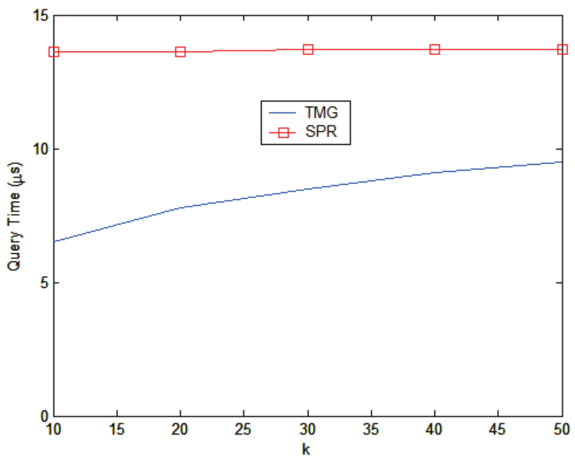
(1) Fig. 6(a). As *k* value increases, SPR needs to sort the winner group to find the highest score *k* data. Obviously, when sorting has been completed, finding 10 data and 50 data makes little difference to the computer. Thus, the time consumed by SPR has barely changed. After multilayer grouping, the data number of our final candidate dataset is much smaller than SPR. Therefore, we consume only 60.6% of SPR's query time. Meanwhile, TMG needs to find the top-k results from more data with the increasing *k* value. Therefore, TMG consumes more query time if *k* value increases.

(2) Fig. 6(b). When |*D*| increases from 1T to 5T, SPR needs to sort the growing number of data of winner set. So the time consumed gradually increases. Through the multilayer grouping, TMG's data number of candidate dataset that ultimately needs to be sorted does not change significantly with the increasing |*D*|. Therefore, TMG's query time growth rate is significantly lower than SPR.

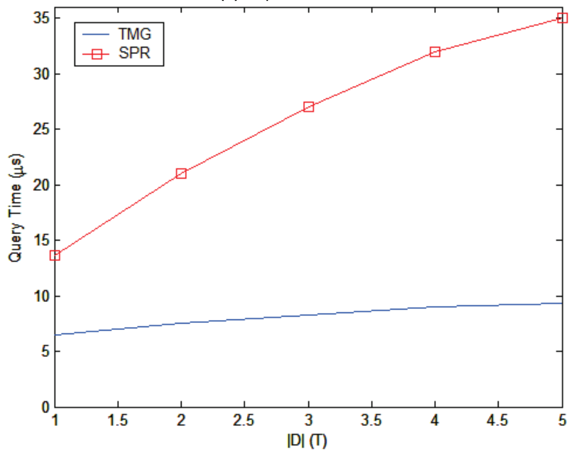
By synthesizing all the experimental data, we can draw the following conclusions.

(1) The comprehensive performance of our TMG is obviously superior to that of the existing SPR under assumption that index is already built.

(2) TMG has a very good performance in the face of various parameters if index is already built. Thus, TMG can be widely used in various application scenarios, such as big data, distributed computing, e-commerce, car networking and so on.



(a) Top-k's kvalue



(b) Number of data in candidate dataset *D*

Figure 6 The comparison of top-k query

5 CONCLUSIONS

Most of existing methods have some or all shortcomings as follows. (1) High sorting costs. (2) Insufficient grouping. (3) High memory consumption. To remove these issues, we propose an efficient Top-K query scheme based on multilayer grouping. By grouping data in multiple layers, the data number of final candidate set for top-k query is close to *k* value. Thus, we almost minimize the sorting cost. Meanwhile, this multilayer grouping method removes almost all redundant data. The experimental results show that we can effectively reduce memory consumption and improve top-k query efficiency in cases when index is already built.

Our data is from crowd sourced pairwise judgments, which is fixed. However, celebrities' scores change with the number of scorers. Therefore, our subsequent research will consider the dynamic changes of celebrities' scores. That is, we will re-group the changed scores to support the dynamic scenarios. On the other hand, in some application scenarios, the number of data in middle set is still too large. Thus, we need to multilayer group the middle set to further enhance the efficiency of top-k query.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (No. 61762055, 61962029, 61462048, 61662039); the Jiangxi Provincial Natural Science Foundation of China (No. 20181BAB202014, 20192ACBL20031); the Humanities and Social Sciences Foundation of Colleges and Universities in Jiangxi Province (No. TQ18111); and the Key Program of Zhejiang Provincial Natural Science Foundation of China (No. LZ18F020001).

6 REFERENCES

[1] Mouratidis, K. & Tang, B. (2018). Exact processing of uncertain top-k queries in multi-criteria settings. *VLDB Endowment, 11*(8), 866-879. https://doi.org/10.14778/3204028.3204031

[2] Zhang, Z., Wei, X., Xie, X., Pan, H., & Miao, Y. (2018). An Efficient Optimization Approach for Top-k Queries on Uncertain Data. *International Journal of Cooperative Information Systems, 27*(01), 1741002. https://doi.org/10.1142/S0218843017410027

[3] Mouratidis, K. (2017). Geometric approaches for top-k queries. *VLDB Endowment, 10*(12), 1985-1987. https://doi.org/10.14778/3137765.3137826

[4] Cui, Z., Li, H., & Zhu, H. (2018). ETGP: Top-K Geography-Text P/S Approach without Threshold. *Technical Gazette, 25*(5), 1408-1413. https://doi.org/10.17559/TV-20180408134120

[5] Zhang, W., Zhang, Z., & Chao, H. C. (2017). Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management. *IEEE Communications Magazine, 55*(12), 60-67. https://doi.org/10.1109/MCOM.2017.1700208

[6] Kou, N. M., Li, Y., Wang, H., Hou, U. L., & Gong, Z. (2017). Crowdsourced Top-k Queries by Confidence-Aware Pairwise Judgments. *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, 1415-1430. https://doi.org/10.1145/3035918.3035953

[7] Dutta, S. & Jacobson, S. H. (2018). Modeling the NCAA basketball tournament selection process using a decision tree. *Journal of Sports Analytics, 4*(1), 65-71. https://doi.org/10.3233/JSA-170149

[8] Shanbhag, A., Pirk, H., & Madden., S. (2018). Efficient Top-k Query Processing on Massively Parallel Hardware. *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1557-1570. https://doi.org/10.1145/3183713.3183735

[9] Deshmane, P., Patil, P., & Pathak, A. (2018). Extraction of Top-k List by Using Web Mining Technique. *Information and Communication Technology for Sustainable Development*, 285-293. https://doi.org/10.1007/978-981-10-3920-1_29

[10] Yang, X., Ajwani, D., Gatterbauer, W., Nicholson, P. K., Riedewald, M., & Sala, A. (2018). Any-k: Anytime Top-k Tree Pattern Retrieval in Labeled Graphs. *Proceedings of the 2018 World Wide Web Conference (WWW)*, 489-498. https://doi.org/10.1145/3178876.3186115

[11] Lapin, M., Hein, M., & Schiele, B. (2018). Analysis and Optimization of Loss Functions for Multiclass, Top-k, and Multilabel Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(7), 1533-1554. https://doi.org/10.1109/TPAMI.2017.2751607

[12] Agarwal, P. K., Kumar, N., Sintos, S., &Suri, S. (2018). Range-max queries on uncertain data. *Journal of Computer and System Sciences*, 94, 118-134.

https://doi.org/10.1016/j.jcss.2017.09.006

[13] Li, Y., Kou, N. M., Wang, H., Hou, U. L., & Gong, Z. (2017). A confidence-aware top-k query processing toolkit on crowdsourcing. *VLDB Endowment, 10*(12), 1909-1912. https://doi.org/10.14778/3137765.3137806

[14] Li, K., Zhang, X., & Li, G. (2018). A Rating-Ranking Method for Crowdsourced Top-k Computation. *Proceedings of the International Conference on Management of Data (SIGMOD)*, 975-990. https://doi.org/10.1145/3183713.3183762

[15] Lee, J., Lee, D., & Hwang, S. W. (2017). CrowdK: Answering top-k queries with crowdsourcing. *Information Sciences*, 399, 98-120. https://doi.org/10.1016/j.ins.2017.03.010

[16] Chen, L., Shang, S., Yao, B., & Zheng, K. (2018). Spatio-temporal top-k term search over sliding window. *World Wide Web*, 1-18. https://doi.org/10.1007/s11280-018-0606-x

Contact information:

Zongmin CUI, Assoc. Prof., PhD
School of Information Science and Technology, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: cuizm01@gmail.com

Yu GAO, BE
School of Information Science and Technology, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: 2366635279@qq.com

Caixue ZHOU, Assoc. Prof.
School of Information Science and Technology, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: charlesjjjx@126.com

Guangyong GAO, Prof., PhD
School of Computer and Software,
Nanjing University of Information Science and Technology,
No. 219, Ningliu Road, Nanjing, Jiangsu 210000, China
School of Information Science and Technology, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: gaoguangyong@163.com

Zhuolin MEI, PhD
(Corresponding author)
School of Information Science and Technology, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: meizhuolin@126.com

Zongda WU, Prof., PhD
Oujiang College, Wenzhou University,
Wenzhou 325035, Zhejiang, China
E-mail: zongda1983@163.com