Cost Optimized Data Access using Rank-Join

Mr. Pravin L. Satore
M E IInd Year Student & Researcher
Department of Computer Science & Engineering
Matoshri Pratishthan Group of Institutions
Nanded, India

Ms. Shital Y. Gaikwad
Assistant Professor
Department of Computer Science & Engineering
Matoshri Pratishthan Group of Institutions
Nanded, India

***Abstract*—The prime task of search computing is to join the result of complex query plans. Join of complex query plan problem is classified in the conventional rank aggregation i.e. combining different ranked lists of objects to produce single valid ranking. Rank-join algorithms provide best overall results without accessing total objects in list. This paper describes further views on topic by emphasizing the study and experiments on algorithms that operate with joining the ranked results produced by search services. The rank-join problem is considered to be extending rank aggregation algorithm to the case of join in setting of relational database. On the other hand search computing join diverges from orthodox relational concepts in many ways. Random and sorted access patterns are used to access the services; accessing service is costly in terms of response time, because usually they are remotely located. The output is returned in pages of answers and criteria is some top-k ranking function; multiple search services to answer the same query, user can also redefine the search criteria. This paper proposes Cost Aware Rank-Join with Random and Sorted Access (CARS) methodology in the context of rank join algorithms for the efficiency of search computing. Experimental results prove that CARS strategy outperforms the existing methods of Data Access in terms of access cost.**

Keywords*— Top-k, Rank-Join, Cost Optimization, Query Optimization, Search Computing, TA Algorithm

INTRODUCTION

Search services uses different types of techniques to rank query answers. Generally, users are only looking for most important query answers, i.e. top-k answers, from bulk of answers. Currently many emerging applications assure that the effective support for top-k queries is there. For example, the success rate of meta-search engines [1] [2] is directly proportional to the use of effective rank aggregation methods. The next challenge is to firmly combine the ranked list of objects and create a single unanimous ranking for the objects. Many applications produce top-k results by joining & aggregating the results from multiple inputs.

Methods in this paper will concentrate on a special kind of top-k processing techniques, i.e. rank-join algorithms [1], which gets the top-k combinations from a data set that comes from joining multiple data sources. These kind of top-k processing techniques are very significant for answering multi-domain queries.

```
SELECT Rank () over (order by h.stars * r.rating) as
          rank, h.hotelname, r.restname, r.street
     FROM    Hotel h, Restaurant r
       WHERE  h.street= r.street
          LIMIT    5
```

Figure 1: Rank-Join Query Example

This involves the answers to be extracted and combined from domain specific search system. Finally, an aggregation function used to form global ranking for every combined answer, so that algorithm can provide answers with top score to user.

The data set produces the output tuples sorted by some score; here the score is certain field of tuples. The ranked list may consist of large number of items represented in pages and cost of accessing these kinds of pages is sometime become intolerable. The objects in the list are retrieved by some methods like sorted *i.e.* resulting a large list of objects ranked by some function, or random *i.e.* resulting a limited set of objects, not ranked but some condition over attribute is fulfilled.

I. ISSUE OF SEARCH COMPUTING

Search computing concentrates on answering complex search queries combining data from several multi-domain search [2] services on web or other platform. These combinations are ranked and joined by some score attached to them. Every combination has a score, usually computed by some aggregation function over scores of every data elements. Mostly users only browse the top answers sorted by score. A simple but effective way is, first fetching the data elements from the data sets, second results are joined to form the combinations, third compute the score of every combinations and finally ordering the combinations by their scores. The basic concepts of rank join algorithms are to explore situation where the input data sets, i.e. relational data table, are already sorted by some score. That is why solution can be proposed to avoid the retrieving of tuples because the top-k combinations of answers can already be formed. The major task then of the conventional rank join algorithms [1], [4], [5] is to optimize or minimize the input/output cost with respect to extremely simple join and sort approach discussed earlier.

Fig. 1 describes a Rank-Join Query example in which two services are used namely Hotel and Restaurant. Hotel service has attributes like HotelName, Location, Street and Stars. The data of this service is ranked by Stars. The other service Restaurant has similar attributes like RestaurantName, Location, Street, and Rating. Restaurants are ranked by Users Rating. In this example the results from each service is filtered, sorted, ranked and joined by using Rank-Join Algorithm. Rank join algorithms are very important aspects of search computing along with that we also need to analyze the individual characteristics search computing especially when operating data sets are dynamic search services rather than relational database tables. In next section we will see an overview of core concepts that elaborate search computing framework [2].

A. *Access patterns*

Search services have some limitation like input to some fields is compulsory to give to get the results. To deal with this kind of restriction, we consider that each service is characterized by a given number of combinations of input and output parameters, called *access patterns*, pointing it to different ways that it can be retrieved. Access pattern is not handled by search computing framework therefore search computing must define an access strategy for its requirements.

B. *Access costs*

Extraction of data is costly and should be handled by effective rank join algorithms i.e. fetching data must be cost optimized. Access cost is partially depends on invoked services and applied access methods. Random and sorted access methods are available methods in context of rank join problem.

Sorted access method retrieves tuples that are sorted by some score and result is open to all search services, but for every new call results in a page of data elements instead of a tuple. In some situation, only one ranking criteria might be present for sorted access to search service. For example, consider a query that intends to find hotels by its stars, like 3 stars or 5 stars, but instead available service can only retrieve the results ordered by its nearest location from airport. In such situation if relation between the stars and location can be mapped then rank join algorithms might find themselves useful to answer such queries.

Random access extracts tuples that relates to a given object, e.g. all the Chinese cuisine restaurants in the city of Mumbai and permits to terminate rank join algorithm early when it is available, and thus narrowing the number of input/output operation. In a relational setting, random access can be provided by building an index on top of one of the attributes of a table. This is not a suitable option when operating in search computing. However, when a search service, say s1,only provides sorted access, it is to some extent possible to obtain random access by invoking another search service, say s2,returning data items of the same kind, although s2 might be characterized by a different access cost and contain only a subset of the data items of s1.

C. *Redundancy of Data Sources*

There may be different search services which can be potentially invoked to answer identical or similar queries. Consider, for example, the excessive number of identical services that search for movies. Such a redundant availability of data sources comes with no additional cost in the context of search computing. If correctly managed, it can be positively used in two ways: one, improving the system response time and second, the robustness to time-varying access costs or service failures. Regarding first goal, parallel invocation of multiple services can be implemented. The availability of multi-domain search services, each referred by its access cost, might provide random access when single service is not able to do so.

D. *Users in the loop*

The queries proposed by users of a search computing system can be shaped at runtime to help satisfying the user's information requirement. The liquid queries paradigm concept portrays a set of operations that can be performed at the client side. Some of these operations do not require communication with the remote search services, since they only influence the visualization of data already available at the client side. Others require extraction of additional data from remote services. For example, the user might want to dynamically adjust the aggregation function. In a weighted sum, this is accomplished by changing the weights assigned to the different search services. In order to preserve the guarantee of displaying the top results, further data might need to be fetched.

- Costs

Cost is associated with each result which when we expect from the search service such that cost of invoking a result. Cost may vary for different services; similarly it also depends on which access pattern we use for fetching the results from search services. We have seen two types of accesses, one is sorted access & other is random access, so does exist cost of access i.e. sorted access cost sc_i and random access cost rc_i [1]. These costs may correspond to the average service response time.

II. RANK-JOIN QUERY PROCESSING

In Top-k join query model, the scores are assumed to be associated to join results rather than base tuples. A top-k join query joins a set of relations based on some subjective join condition, assigns scores to join results based on some scoring function, and provides the top-k join results. A rank-join algorithm implementation is given in [1] [9].

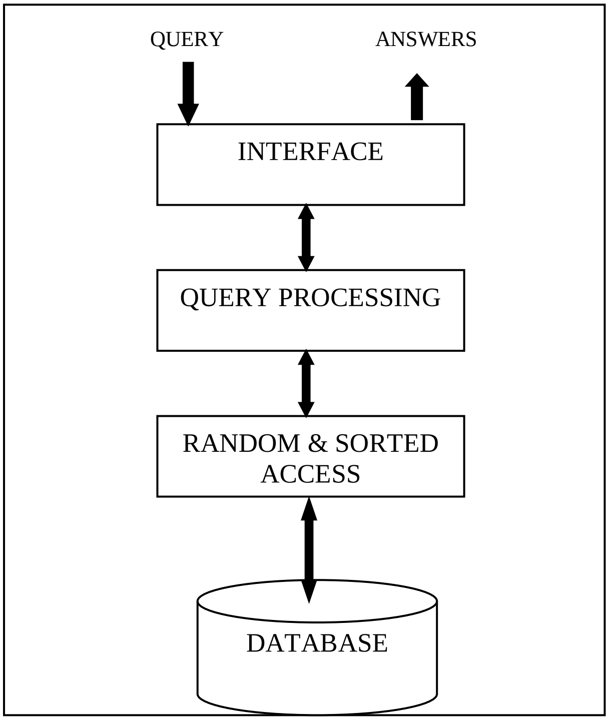


Figure 2: Rank-Join Query Processing

Fig. 2 describes the working of execution and work flow of Rank-Join query. First through interface the query is taken as input then the processing on that query begins where appropriate pulling strategy is applied for accessing data. In this case cost optimized random and sorted access is applied which uses Rank-Join Algorithm. Finally after all processing the top answers are fetched and reported back to the user.

A. *Overview of Rank-join Algorithm*

The Rank-join algorithm work on four tuples (R1, R2, S, k) where R1 and R2 are the two relations located in different database and accessed with sorted access, in decreasing order of S, and random access, based on an input join attribute value. S is a scoring function scoring function upon which the join results are being ranked. k is value between 1 and total number of join result of R1 and R2 i.e. top-k answers and k is any positive integer. A solution is an ordered relation O containing the top-k combinations from R1 join R2 ordered by S.

Now the outline for the Rank-Join Algorithm as follows. The input for Rank-Join Algorithm is two relations R1 and R2. S is the scoring function. The result size is. k. The most likely output will be the top-k combination from R1 and R2 with highest aggregate score. The data necessary will be buffers i.e. P1, P2, RB1, RB2 and O. The P1 and P2 are buffers to hold the data from two relations R1 and R2 respectively. The RB1 and RB2 buffers will be required to hold the sorted and filtered results with upper bound and lower bound from two relations R1 and R2 respectively. A solution is an ordered relation O containing the top-k combinations from R1 join R2 ordered by S.

At beginning the upper bound is not known and the buffers, P1, P2, RB1 and RB2, will be null. Retrieve tuples from R1 and R2 and sort them in descending order of their individual score. For every new retrieved tuple following operations will be performed. First, new valid join combination between tuples of both relations is produced. Second, for every new join combination compute the score by using some predefined score aggregation function. The algorithm preserves a threshold T bounding the scores of join results that are not found yet. The top-k join results are acquired when the minimum score of the k join results with the maximum score values is not below threshold T. the Rank-Join algorithm maintains the scores of the completely seen join combinations only. As a result, the Rank-Join algorithm reports the exact scores of the top-k tuples. This procedure will continue till new combination that has a score of exactly equal to the lower bound or given limit is found. As soon as this happens the algorithm stops and gives the top-k combination as final result.

B. *Rank-Join Query plan*

Fig. 3 depicts the query plan generated by Rank-Join Algorithm. The join expression is a kind of a rank join operator.

Rank operators pipeline their outputs by upper bounding the scores of their unseen inputs, allowing for consuming a small number of tuples in order to find the top-k query results. Rank operators need to be integrated with query optimizers to be practically useful. Top-*k* queries often involve different relational operations such as joins, selections and aggregations. Building a query optimizer that generates efficient query plans satisfying the requirements of such operations, as well as the query ranking requirements, is vital for efficient processing.

An observation that encourages the need for integrating rank operators within query optimizers is that using a rank operator may not be always the best way to produce the required ranked results.

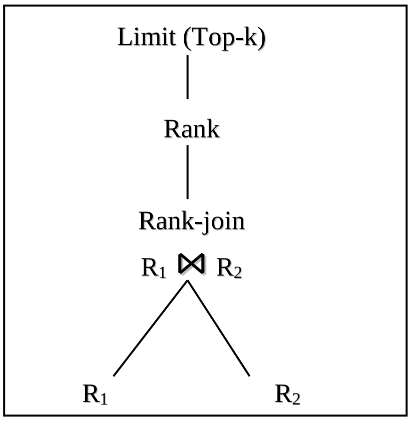


Figure 3: Rank-Join Query Plan

C. *HRJN*

A two-way hash join implementation of the Rank-Join algorithm, known as Hash Rank Join Operator (HRJN), is presented in [9]. HRJN is based on symmetrical hash join. HRJN operator [9] keeps a hash table for each relation that is in the process of join, and a priority queue is also maintained to buffer the join results in the order of their scores. The hash tables contains input tuples seen so far and are used to compute the valid join answers. The HRJN operator implements the old-fashioned iterator interface of query operators, which comprise two methods: *Open* and *GetNext* [9]. The Open method is liable for initializing the necessary data structure; the priority queue Q, and the left and right hash tables.

In [3], an enhancement of HRJN algorithm is provided where a dissimilar bounding scheme is used to compute the threshold *T*. This is accomplished by computing a *feasible region* in which unseen objects may exist. Feasible region is computed upon the objects seen so far, and knowing the possible range of score predicates. The algorithm reports the next top join result as soon as the join result at queue top includes an object from each ranked input.

III. COST OPTIMIZED APPROACH

This paper introduce CARS (**C**ost **A**ware rank join with **R**andom and **S**orted access), a pulling strategy described at compile time that takes into account access costs. The pulling strategy is attained by solving an optimization problem that seeks to minimize the cost incurred by Rank-Join Algorithm to find a target number of top combinations. Formally, following problem is resolved:

Minimize $\overline{C}(n_1, n_2)$

Subject to $\overline{K}(n_1, n_2) \geq K_T$ with $n_i \in N \cap [0, N_i]$ …. (1)

Where $\overline{K}(n_1,n_2)$ denotes the expected number of combinations that can be formed by retrieving n_i tuples from service s_i by means of sorted accesses only, $\overline{C}(n_1,n_2)$ is the related likely cost of execution of both sorted and random accesses, i.e., those desired to find the top-k combinations according to Rank Join Algorithm, and K_T is a target number of combinations. Both prospects are taken with respect to all the possible ways of composing the tuples returned by the services. Note that problem/equation (4) does not constrain the number of top combinations directly, but rather the expected number $\overline{K}$ of combinations constructible using sorted access only. These may be taken as good combinations since, being fetched by sorted access; they have high scores both for s_1 and the s_2 sub tuples. The rationale behind this choice is that

This optimization does not require any information about the score distribution, which might be difficult to obtain;
When K good combinations are found after sorted access by Rank Join Algorithm at least K top combinations (which include all the K good combinations) will be formed with random access.
Similarly, solve the problem of maximizing the number of top-k combination to be found, as:

Maximize $\overline{K}(n_1,n_2)$

Subject to $\overline{C}(n_1,n_2) \le C_T$, with $n_i \in N \cap [0, N_i]$ …. (2)

Where, $\overline{K}$ and $\overline{C}$ are as declared before and C_T is the target cost.

A. Cost constraint formula

In order to find the top-k combinations, now make random accesses. For that it is essential to retrieve the tuples in s2 whose join value appeared at least once in the first n1 tuples of s1, and vice versa. Top combinations are essentially included in the union of the combinations formed after the sorted accesses and random accesses.

In the following, assume that the cost of retrieving the tuples dominates over the cost of computing the combinations and their scores [8]. This is sensible in this context, since services are typically accessed remotely on the web. This Paper implements the following additive cost model:

$$C(n_1,n_2) = sc_1\, n_1 + sc_2\, n_2 + rc_2\, j_1(n_1) + rc_1\, j_2(n_2) \text{ ...... (3)}$$

Where sc_i is the unitary sorted access cost (per tuple),
rc_i is the unitary random access cost (per distinct join attribute tuple), and
$j_i(n_i)$ is the number of distinct join attribute tuples retrieved from s_i.

Note that the random access cost of s_1 rest on on rc_i and on $j_i(n_i)$, and vice versa.
In order to solve the query optimization problem formulated above, which refers to a single rank-join operator on two services, CARS described a cost model that illustrates the cost function C(n1, n2). Most of the previous literature on rank-join adopts a simple additive model, whereby the cost is defined as the sum of the costs of all I/O operations. Both sorted and random access (whenever available) costs need to be taken into account, meanwhile they are possibly characterized by heterogeneous costs, because of the fact that random accesses might potentially refer to data that is stored in other external data sources. While this approach is still applicable in the context of search computing, we want to take benefit of the fact that services are typically available at remote servers. Therefore, more flexibility is given in the way services can be invoked, i.e. by exploiting parallel invocation. Nevertheless, parallelism also influences both the actual execution strategy and the cost model that drives the query optimization and these problems are carefully addressed in ours cost model.

IV. EXPERIMENTS

Examine the overall cost to return the top-k combination when applied Rank-Join search that has CARS (**C**ost **A**ware **R**andom and **S**orted) Access which uses the Rank-Join Algorithm. This experiment conducts analysis on real data sets. For now assess the impact of parameters *i.e.* Top-k Combinations, Score Distribution and Overall Cost.

Data Sets: Firstly consider two services which provide both sorted and random access. The first service is "hotel" for hotels as s_1 and "restaurant" for restaurants as service s_2.
The service and available access patterns lets us requesting hotels by sorted access ranked by stars in descending order from luxury hotels down to hotels with no stars, and by random access by searching for hotels in a given street.
Results are paginated with page size P_1=23 tuples/page. The total number of tuples of s_1 is N_1=516, the number of distinct join values *i.e.* different streets is J_1=186. The average number of hotels per street is Q_1=2.77.
The service s_2 can be invoked by sorted access returning restaurants ranked by customer rating in the [0, 10] range and by random access by searching for restaurants in a given street.

Search results are paginated with a page size P_2=20 tuples/page. The total number of tuples of s_2 is N_2=509, the number of J_2=171. The average number of restaurants per street is Q_2=2.97.
Methods: Test two different pulling strategies applied to rank join algorithm, endowed with both random and sorted access:

A. Round Robin

It's alternating Sorted Access to s1 and s2. This pulling strategy is well-defined by "HRJN" [9] and Fagin's "Combined Algorithm" as regards sorted access. Fagin's "Combined Algorithm" forces to perform all random access. When two services are characterized by dissimilar page sizes, the round robin strategy is adapted to select the service with the least depth, so as to make sure that both services are explored up to same depth.

B. Score-Aware (SA)

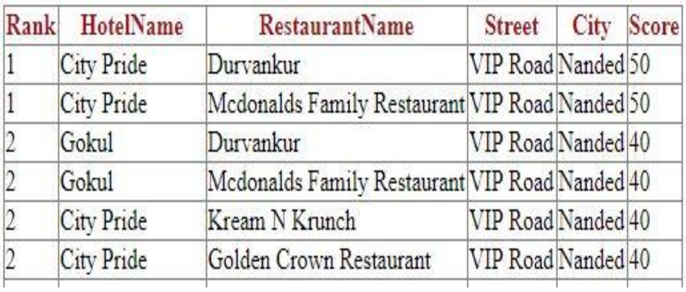
Score-Aware strategy [7] that decides which service to access next based on the scores of the retrieved tuples by comparing their bounds. This strategy is used in HRJN*[8].

SA produces better results but takes much more time i.e. Cost is higher in compare with Round Robin.

C. Cost-Aware with Random and Sorted Access (CARS):

CARS is the pulling strategy that defined in previous section. In this particular data access method aim is to minimize the COST at the same time maximizing the Top-k combinations to be found.

When given the input of P1 23 and P2 20, CARS along with Rank-Join algorithm is used to generate the results then to find the top 100 hotel restaurant combination in just 15 miliseconds. In comaparison with the results of Round Robin Strategy, Round Robin takes 39 miliseconds.



Rank	HotelName	RestaurantName	Street	City	Score
1	City Pride	Durvankur	VIP Road	Nanded	50
1	City Pride	Mcdonalds Family Restaurant	VIP Road	Nanded	50
2	Gokul	Durvankur	VIP Road	Nanded	40
2	Gokul	Mcdonalds Family Restaurant	VIP Road	Nanded	40
2	City Pride	Kream N Krunch	VIP Road	Nanded	40
2	City Pride	Golden Crown Restaurant	VIP Road	Nanded	40

Figure 4. Top-k hotel restaurant combination on given street

- Result Analysis

The Results obtained on the given datasets are shown in fig 4**.** As CARS strategy generates way better results than Round Robin method. Now in the following section overlook the individual parameters i.e. page size, top-k combinations found, score distribution and cost.

Page Size – page size is individual & one kind of input parameter. Search services usually return the results in pages, the number of results on each page may be large sometimes. For that reason characterize the search service result with a page size Pi.

Top-k combinations found – Fig. 5 shows the result of CARS method and Round Robin method when required to find top-k hotel restaurants. As given in figure when given the input of P1=23 and P2=20, CARS is able to find top-100 combinations where Round Robin is only able to find 25 combinations. This Experiment has tested the result with many different inputs still CARS produces better results in all cases

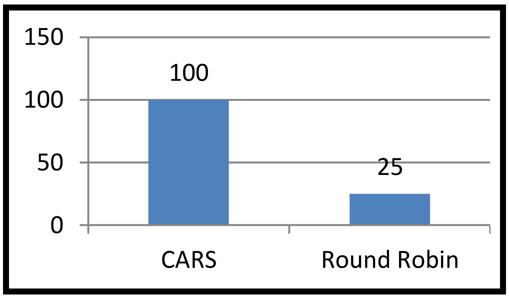


Figure 5: Top-k combination found

Score Distribution – Fig. 6 depicts the results of score distributions of CARS and Round Robin method. In score distribution the maximum score is actually the score of the top most combination and minimum score is score of low most combination. The formula to calculate the score is Hotel star * Restaurant rating. The score distribution of CARS methodology is way better than Round Robin. The maximum score generated by CARS is 50; this is highest score that can generate in any case using any methodology. The minimum score generated by CARS is 28. Where, Round Robin method generates the maximum score of 25 and minimum score of 7. Though given the different inputs CARS is able to produce long range of score distribution. So CARS method has long range to produce the top-k combination while Round Robin, in comparison, has very narrow range to produce the top-k combination.

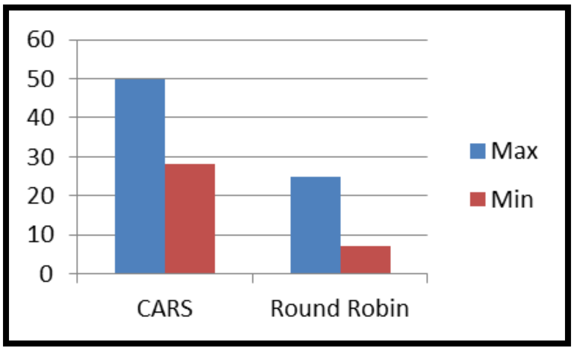


Figure 6: Score Distribution

Cost – Cost is the most important and algorithm defining factor here, upon this cost parameter experiment is able to prove which methodology is better.

As in fig. 7 we can observe that, after the execution the Round Robin method takes 44 MS to execute and produce the result. While on the other hand CARS methodology which uses Rank-Join algorithm is able to produce the required top-k combination in 13 MS. So in comparison CARS method is way ahead of Round Robin.

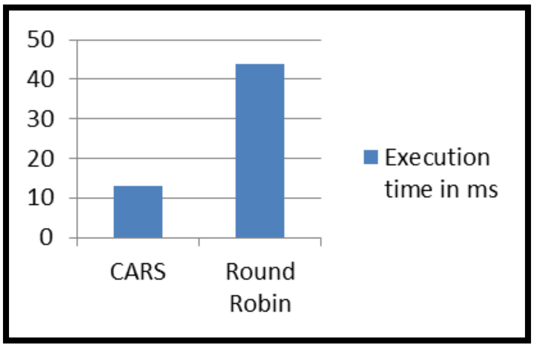


Figure 6: Overall Cost

From the results of CARS methodology and the performance analysis seen in the previous figures CARS Method has produced a cost optimized strategy that uses rank-join algorithm. The performance of CARS method is cost optimized.

V. CONCLUSION

Encouraged by the goal of answering multi-domain queries with cost optimization propose an execution strategy in this paper, which retrieves top-k combinations that can be formed by joining the results of heterogeneous search services. By using random and sorted access this paper have defined optimized cost aware strategy with an additive cost model.

This paper have successfully implemented Rank-Join algorithm to achieve optimality in terms of cost as well as accuracy by using both access methods i.e. random and sorted data access.

In the future work, the query optimization framework can be extended to the non-additive cost model that will access the services in parallel along with pipelining the joins.

ACKNOWLEDGMENT

I am great thankful to Director, Dean, HOD, Guide, staff members of CSE Department from Matoshri Pratishthan Group of Institutions, Nanded and finally The Guide Ms. Shital Y. Gaikwad for her guidance and moral support in research and publication of this article.

REFERENCES

[1] Davide Martinenghi, Marco Tagliasacchi, “Cost-Aware Rank Join with Random and Sorted Access,” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 12, DECEMBER 2012.

[2] Search Computing - Challenges and Directions, M. Brambilla and S. Ceri, eds. Springer, Mar. 2010.

[3] Karl Schnaitter, Neoklis Polyzotis, “Evaluating Rank Joins with Optimal Cost,” ACM 978-1-60558-108-8/08/06, Vancouver, BC, Canada, 2008.

[4] Christian A. Lang, Yuan-Chi Chang, John R. Smith, “Making the Threshold Algorithm Access Cost Aware” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 16, NO. 10, OCTOBER 2004.

[5] Jonathan Finger, Neoklis Polyzotis, “Robust and Efficient Algorithms for Rank Join Evaluation” SIGMOD’09, June 29–July 2, 2009, Providence, Rhode Island, USA, ACM 978-1-60558-551-2/09/06, 2009.

[6] IHAB F. ILYAS, GEORGE BESKALES, and MOHAMED A. SOLIMAN, “A Survey of Top-k Query Processing Techniques in Relational Database Systems”, ACM Comput. Surv. 40, 4, Article 11, ACM 0360-0300/2008/10-ART11, October 2008.

[7] J. Finger and N. Polyzotis, “Robust and Efficient Algorithms for Rank Join Evaluation,” Proc. 35th ACM SIGMOD Int’l Conf. Management of Data, pp. 415-428,2009

[8] K. Schnaitter, J. Spiegel, and N. Polyzotis, “Depth Estimation for Ranking Query Optimization,” Proc. ACM SIGMOD Int’l Conf. Management of Data (VLDB), pp. 902-913, 2007.

[9] I.F. Ilyas, W.G. Aref, and A.K. Elmagarmid, “Supporting Top-k Join Queries in Relational Databases,” VLDB J., vol. 13, no. 3, pp. 207-221, 2004.

AUTHORS PROFILE

Mr. Pravin L Satore has received the B. Tech., degree in Computer Science and Engineering from SGGS Institute of Engineering and Technology, Nanded, Maharashtra, India in 2009. During 2009-11 he worked as Asst. Professor SGGS Institute of Engineering and Technology, Nanded, Maharashtra. He is currently pursuing Master of Engineering in Computer Science and Engineering with dissertation topic on Cost Optimized Data Access using Rank-Join. His area of interest for research is Programming languages, Data Structures, Database management system, Query processing and Optimization, ranking queries.

Ms. Shital Y. Gaikwad has received the B.E. and M.Tech degree in Computer Science and Engineering. She is Google scholar, Guided PG students for research work. She has published 3 National Journals and 27 international journals. She has attended and presented papers in National and International conferences, She has attended Industrial training and Faculty deveopment programs. She is awarded with Dr.APJ Abdul Kalam life time achievement National award,Bengaluru 2018 and Rotary Vocational Service Excellence Award. Mimansa award 2018.