Procedia Computer Science
Volume 80, 2016, Pages 2211–2215
ICCS 2016. The International Conference on Computational Science

Efficient Skyline Query over Multiple Relations

Jinchao Zhang[1,2], Zheng Lin(✉)[1], Bo Li[1], Weiping Wang[1], and Dan Meng[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{zhangjinchao,linzheng,libo,wangweiping,mengdan}@iie.ac.cn

Abstract
Skyline query on multiple relations, known as skyline join query, finds skyline points from join results of multiple data sources. The issue of skyline join query has been extensively studied. However, most of the existing skyline join algorithms can only perform query on two relations, and ignores the common occasion which involves more than two relations. In this paper, we propose an efficient skyline join algorithm *Skyjog*, which is applicable for query on two or more relations. *Skyjog* can quickly identify most of skyline join results with simple calculation. Extensive experiments demonstrate that *Skyjog* outperforms the state-of-the-art skyline join algorithms on two and more than two relations.

Keywords: Skyline Query, Group Division, Multiple Relations

1 Introduction

Skyline query aims to find interesting points from the given dataset. For example, a tourist traveling to the seaside is looking for a hotel that is not only cheap but also close to the beach[1]. The process of finding interesting hotels is skyline query, and the price and distance are regarded as *skyline attributes*.

In some applications, *skyline attributes* belong to multiple relations, and the issue of computing skyline results on multiple relations is termed as *skyline join* query. There are many *skyline join* algorithms, while most of them are limit to two relations. Though two algorithms $S^2J\text{-}M$ [7] and $S^3J\text{-}M$ [7] have been proposed for *skyline join* on more than two relations, they are just the simple extension of existing two-relation skyline join algorithms.

In this paper, we study the issue of *skyline join* query over multiple relations, and propose an efficient *skyline join* algorithm *Skyjog* based on group division approach. For each relation, tuples are grouped according to the join attribute. *Skyjog* divides these tuples into several partitions depending on the dominance relationships of inter-group and intra-group. Based on the properties of group division, tuples generated by some join combinations are guaranteed to be skyline points. To obtain the final result, *Skyjog* only has to check tuples that are generated by other join combinations. Benefiting from the group division approach, *Skyjog* efficiently reduces the size of intermediate results and avoids much redundant computation.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.381

2 Related Work

Some of the prior work on *skyline join* query include [3],[4], [5],[8],[2],[6] and [7]. Although these algorithms compute skyline results using different approaches and techniques, they try to improve performance by avoiding redundant computation and reducing intermediate datasets. In the following, we review some representative algorithms.

The *skyline join* query was firstly studied by Jin et al. [3]. Jin et al. [4] developed non-blocking algorithms for skyline queries on equi-joins. Vlachou et al. [8] proposed an algorithm *SFSJ* which is inspired by *SFS* algorithm [2]. Nagendra and Candan [6] introduced a concept of layer/region pruning (*LR-pruning*) for *skyline join* queries. Based on this pruning approach, the authors proposed S^2J algorithm and S^3J algorithm in the same literature. Base on the work [6], Nagendra and Candan proposed two algorithms S^2J-M and S^3J-M in [7], which are the first and the only existing algorithms for *skyline join* on more than two relations.

3 Preliminaries

Let R denote a relation, and B denote the attribute set of R. Let τ and τ' denote tuples in R. We can claim that the tuple τ' is dominated by τ, denoted as $\tau \prec_B \tau'$, if the following conditions are satisfied simultaneously, 1) $\tau.a_i \leq \tau'.a_i$[1], 2) $\exists a_j, \tau.a_j < \tau'.a_j$, where $a_i, a_j \in B$, and $\tau.a_i$ denotes the value of τ on the attribute a_i. The skyline query of R with respect to B is defined as $SKY_B(R)= \{\tau_t |\ \nexists \tau_k \text{ s.t. } \tau_k \prec_B \tau_t, \tau_t, \tau_k \in R\}$. The attributes in B are termed as *skyline attributes*. For simplicity, we use *SKY(R)* instead of $SKY_B(R)$ in this paper.

For *skyline join* query, the *skyline attributes* are distributed over multiple relations. For example, for the query on M relations $R_1, R_2, ..., R_M$, the set of *skyline attributes* consists of M disjoint sets, $B_1, B_2, ..., B_M$, where B_i is the *skyline attribute* set of R_i. Intuitively, the result can be calculated by performing skyline query on the join result of these M relations, which is $SKY_{B_1 \cup B_2 \cup ... \cup B_M}(R_1 \bowtie R_2 \bowtie ... \bowtie R_M)$[2]. However, such an approach of calculating skyline join result is inefficient, since the join operation of M relations leads to a large intermediate dataset.

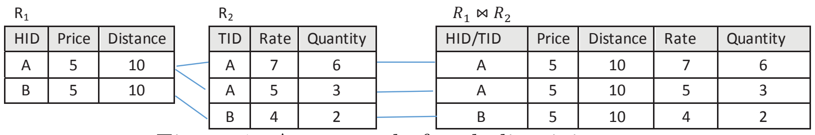
As stated in [3], for *skyline join* on two relations, each relation is partitioned into groups in terms of the join attribute, i.e., tuple τ and tuple $\tau\prime$ are in the same group if the join attribute of them are the same. We claim that τ is a local skyline point if it is not dominated by other tuples of its group, and τ is a global skyline point if it is not dominated by other tuples of all groups. Tuple τ belongs to one of three cases[3], a) *LSS*, τ is both a local skyline point and a global skyline point. b) *LSN*, τ is a local skyline point, but not a global skyline point. c) *LNN*, τ is not a local skyline.

4 Our Solution

4.1 The Amendment on Previous Work

Let *LSS*(R_i) denote the *LSS* tuples of relation R_i, and *LSN*(R_i) denote *LSN* tuples of relation R_i. The existing *skyline join* query algorithms in [3] and [8] are based on the property of $LSS(R_1) \bowtie LSS(R_2) \subseteq SKY(R_1 \bowtie R_2)$ (property 1) and $LSS(R_1) \bowtie LSN(R_2) \subseteq SKY(R_1 \bowtie R_2)$ (property 2), where R_1 and R_2 are joinable relations. However, we find that *property 2* does not always hold. As shown in Figure 1, the *HID* and *TID* are join attributes for R_1 and R_2 respectively, and the rest are *skyline attributes*. We note that the tuple *(A,5,10)* belongs to *LSS*(R_1) , and the tuple *(A,5,3)* belongs to the *LSN*(R_2), but *(A,5,10)* · *(A,5,3)*[4] is not a skyline tuple, since it is dominated by *(B,5,10)* · *(B,4,2)*.

[1]Small values are preferable in this paper.
[2]Join referred in this paper indicates equi-join operation.
[3]*LSS, LSN* and *LNN* are denoted as *LS(S), LS(N)* and *LN(N)* in original paper.
[4]The symbol '·' is the join operator for two tuples.



R_1

HID	Price	Distance
A	5	10
B	5	10

R_2

TID	Rate	Quantity
A	7	6
A	5	3
B	4	2

$R_1 \bowtie R_2$

HID/TID	Price	Distance	Rate	Quantity
A	5	10	7	6
A	5	10	5	3
B	5	10	4	2

Figure 1: An example for skyline join query

The *property 2* holds if we put a restriction on relations, *the vector consisting of all skyline attributes of the tuple is unique in inter-group* , i.e., for a tuple τ in group A, then there does not exist a tuple $\tau\prime$ in other groups that $\tau.a_i$ is equal to $\tau\prime.a_i$, for any $a_i \in B$. The following statements and datasets in experiments comply with this restriction.

4.2 Foundation of Our Algorithm

The two properties (*property 1* and *property 2*) make it possible for identifying a portion of *skyline join* results before join operation. For *skyline join* on two relations, R_1 and R_2, each relation is partitioned into two groups, *LSS* and *LSN* (*LNN* is pruned, see Section 3). Tuples generated by three out of four join combinations, $LSS(R_1) \bowtie LSS(R_2)$, $LSS(R_1) \bowtie LSN(R_2)$, and $LSN(R_1) \bowtie LSS(R_2)$, are guaranteed to be skyline points. We only have to examine tuples in $LSN(R_1) \bowtie LSN(R_2)$, then the results of *skyline join* on R_1 and R_2 are obtained.

However, employing the group division approach for *skyline join* on more than two relations is a bit complex. For *skyline join* on M relations, R_1 and R_M are head relation and rear relation respectively, they only have the join relationships with one relation. The join attribute of R_1 is termed as *<u>r</u>ight <u>j</u>oin attribute(rj)*. Similarly, the join attribute of R_M is termed as *<u>l</u>eft <u>j</u>oin attribute(lj)*. Each of other relations has both *lj* attribute and *rj* attribute simultaneously.

In order to utilize the group division approach, relations have to be partitioned in terms of a join attribute, this join attribute is termed as division attribute(DA). The join attribute used for current join operation is chosen as DA. The *skyline join* on M relations is shown in Figure 2. In each step, the first two relations are selected as the left operand and right operand for join operation respectively. Then the intermediate result is served as left operand in next processing step. We notice that all relations excluding R_1 serve as the right operand in join operation. Thus, these relations use *lj* attribute as DA, only R_1 uses *rj* attribute for group division. Similar to *skyline join* on two relations, *LNN* of the head relation and rear relation are pruned, since tuples in *LNN* cannot contribute to the final results as well. However, tuples of *LNN* in other relations have an opportunity to generate a skyline point.

The left operand (denoted as L) contains two parts *LSS* and *LSN*, while the right operand (denoted as R) has one more part *LNN*, if this operand is not the rear relation. Tuples generated by $LSS(L) \bowtie LSS(R)$, $LSS(L) \bowtie LSN(R)$, and $LSN(L) \bowtie LSS(R)$ are still guaranteed to be skyline points in $L \bowtie R$, and belong to $LSN(L \bowtie R)$. Tuples generated by other join combinations, $LSS(L) \bowtie LNN(R)$, $LSN(L) \bowtie LSN(R)$ and $LSN(L) \bowtie LNN(R)$, have to be determine further, and we use U to denote the set of all these uncertained tuples. If tuple t in U belongs $LNN(L \bowtie R)$, then it will be discarded. Otherwise, t will be put in partition of $LSS(L \bowtie R)$ or $LSN(L \bowtie R)$, depending on if t is a global skyline point or not. Then $LSS(L \bowtie R) \cup LSN(L \bowtie R)$ will be the left operand in next step.

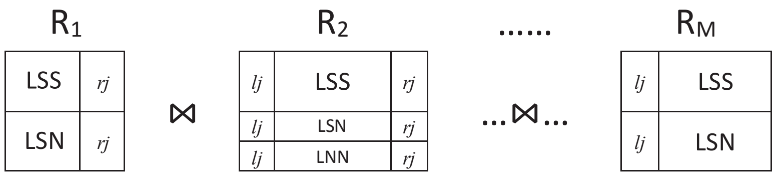


Figure 2: Skyline join query on multiple relations

5 Performance Evaluation

Experiments are conducted on a Linux server with a 2.2 GHz CPU, 32GB RAM. We modify the tool written by Börzsöny [1] to generate experimental datasets. The datasets are generated based on independent, correlated and anti-correlated distributions. The number of relations involved in each query varies from 2 to 4. The join rate[5] ranges from 0.1 to 0.5. The cardinality of each dataset is 100k.

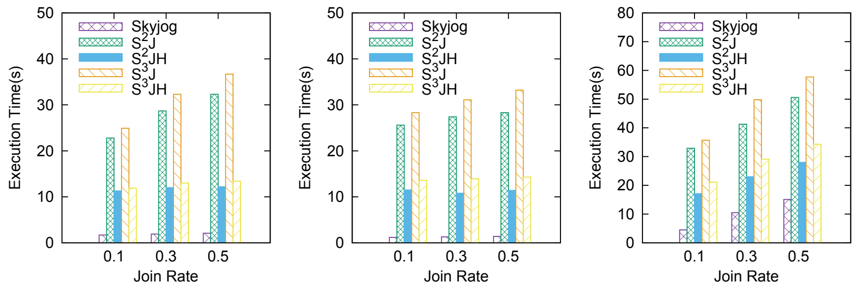
We use the state-of-the-art algorithms for performance comparison. They are S^2J and S^3J for *skyline join* on two relations, $S^2J\text{-}M$ and $S^3J\text{-}M$ for *skyline join* on more than two relations. These four algorithms employ the B-tree index for join operation. However, we find that B-tree is not efficient. Thus, we improve these algorithms by utilizing hash approach for join operation, and name them as S^2JH, S^3JH, $S^2J\text{-}MH$ and $S^3J\text{-}MH$ respectively.

5.1 Evaluation of queries on two relations

Experiments are carried out on two 3-dimensional relations, and the two relations have the same distribution. The execution time of the algorithms is shown in Figure 3.

As expected, the *Skyjog* algorithm outperforms other alternatives on all datasets. It is about one order of magnitude faster than S^2J and S^3J on independent and correlated distribution (Figure 3(a) and Figure 3(b)), and is about 3 times faster than S^2J and S^3J on anti-correlated distribution (Figure 3(c)). This result demonstrates that the group division approach utilized by our algorithm has an remarkable effect on performance promotion, since group division can avoid much point comparison, and it also reduces the number of intermediate tuples generated by join operation.

The S^2JH algorithm and S^3JH algorithm result in about 50 percent performance improvement compared to S^2J and S^3J, and this can be explained that the average search time for hash index is O(1), while that is O(log(n)) for the B-tree index. Thus, S^2JH and S^3JH are more efficient.



(a) independent distribution (b) correlated distribution (c) anti-correlated distribution

Figure 3: Skyline join query on 2 relations

5.2 Evaluation of queries on more than two relations

In this subsection, we evaluate the algorithms on 4 relations, and use only one join rate 0.1 in this experiment. The execution time of the algorithms is shown in Figure 4.

We can see that *Skyjog* outperforms other competitive algorithms on all datasets, and the algorithms with the hash index are superior to corresponding algorithms with the B-tree index. To be specific, the execution time of *Skyjog* barely changes with various dimensions. However, the execution time of other algorithms rapidly increases with the growth of dimensions. The largest performance gap between *Skyjog* and others occurs on datasets with 4 dimensions, as

[5] Join rate is the proportion of tuples in a dataset that will be involved in join result.

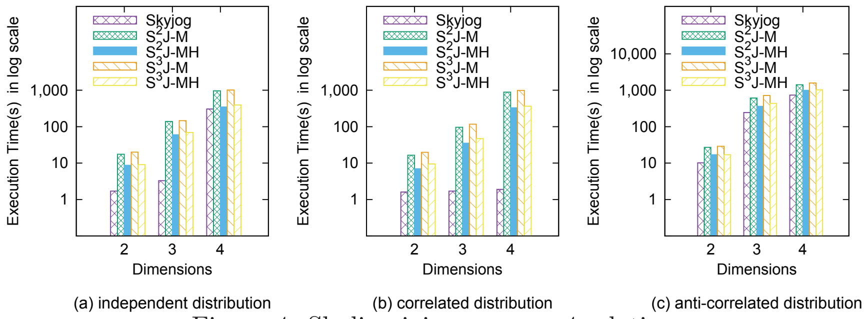


Figure 4: Skyline join query on 4 relations

shown in Figure 4(b). *Skyjog* is about two orders of magnitude faster than other algorithms. This can be explained that group division prunes most of the points before join operation, while $S^2J-M(H)$ has to perform the join operation on all outer relations(see [7] for details) without any point pruning. On independent and anti-correlated distributions, the execution time of all algorithms increases as the the number of dimensions varies from 2 to 4. Since the cardinality of skyline results rapidly increases with the growth of dimensions, and the group division used by *Skyjog* starts being less effective.

6 Conclusion

This paper studies the issue of *skyline join* query processing. We propose an efficient algorithm *Skyjog*, which performs *skyline join* on two or more relations. *Skyjog* is based on the group division approach, and achieves a significant performance improvement by reducing the size of intermediate results and avoiding redundant computation. Extensive experiments demonstrate that *Skyjog* performs well on various datasets, and outperforms the state-of-the-art algorithms on all experimental datasets.

Acknowledgments

This work is supported by the National KeJiZhiCheng Project (2012BAH46B03), the National HeGaoJi Project (2013ZX01039-002-001-001), the National Natural Science Foundation of China (61502478), and "Strategic Priority Research Program" of the CAS (XDA06030200).

References

[1] S Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001.

[2] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 717–719. IEEE, 2003.

[3] Wen Jin, Martin Ester, Zengjian Hu, and Jiawei Han. The multi-relational skyline operator. In *ICDE 2007*, pages 1276–1280. IEEE, 2007.

[4] Wen Jin, Michael D Morse, Jignesh M Patel, Martin Ester, and Zengjian Hu. Evaluating skylines in the presence of equijoins. In *ICDE 2010*, pages 249–260. IEEE, 2010.

[5] Mohamed E Khalefa, Mohamed F Mokbel, and Justin J Levandoski. Prefjoin: An efficient preference-aware join operator. In *ICDE*, pages 995–1006. IEEE, 2011.

[6] Mithila Nagendra and K Selçuk Candan. Skyline-sensitive joins with lr-pruning. In *Proceedings of the 15th international conference on extending database technology*, pages 252–263. ACM, 2012.

[7] Mithila Nagendra and K Selçuk Candan. Efficient processing of skyline-join queries over multiple data sources. *ACM Transactions on Database Systems (TODS)*, 40(2):10, 2015.

[8] Akrivi Vlachou, Christos Doulkeridis, and Neoklis Polyzotis. Skyline query processing over joins. In *Proceedings SIGMOD 2011*, pages 73–84. ACM, 2011.