SkyCube-Tree Based Group-by Query Processing in OLAP Skyline Cubes

Hideki Sato and Takayuki Usami

Abstract *SkyCube*-tree has been developed to realize efficient range query processing for Skyline Cube (SC). Apart from range queries, this paper demonstrates that *SkyCube*-tree can be made use of efficient group-by query processing, though it is originally designed to realize efficient range query processing. Since a group-by query for SC includes the entire dataset as its processing range, the query processing time is potentially large. From the experimental evaluation, the followings are clarified:

- The size of *SkyCube*-trees is sufficiently allowable, since it is at most 2.5 times as large as that of materialized view.
- The time of *SkyCube*-tree based sequential processing is nearly equal to that of materialized view based one, regardless of its dedication to range query processing.
- The time of *SkyCube*-tree based parallel processing is comparatively small and stable. Even though *cell-granularity* is over 80 %, its processing time is around 10 % of that of materialized view based one.

Keywords Aggregate function · *Skyline* operator · Skyline cube · Group-by query · *SkyCube*-tree · GPGPU

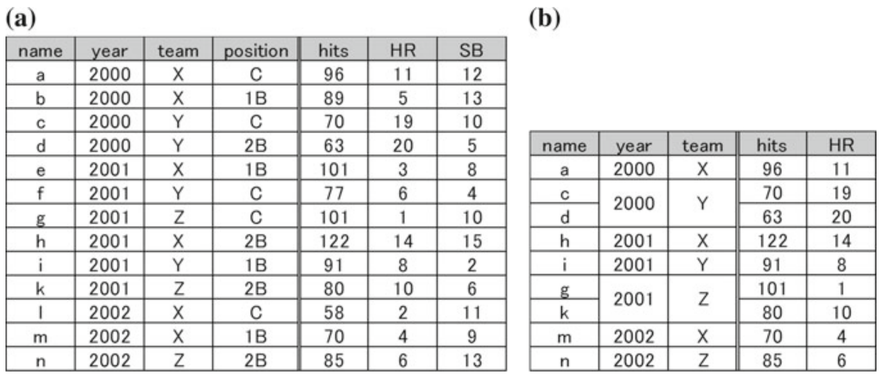
1 Introduction

In Data WareHouse (DWH) environments, On-Line Analytical Processing (OLAP) [1] tools have been extensively used for a wide range of decision-support applications. These tools are built upon Data Cube [2], a collection of data cuboids which

H. Sato (✉)
School of Informatics, Daido University, 10-3 Takiharu-cho, Minami-ku, Nagoya 457-8530, Japan
e-mail: hsato@daido-it.ac.jp

T. Usami
System Development Department, Soft Valley Corporation, Hachioji, Japan

G. De Pietro et al. (eds.), *Intelligent Interactive Multimedia Systems and Services 2016*, Smart Innovation, Systems and Technologies 55,
DOI 10.1007/978-3-319-39345-2_26



(a)

name	year	team	position	hits	HR	SB
a	2000	X	C	96	11	12
b	2000	X	1B	89	5	13
c	2000	Y	C	70	19	10
d	2000	Y	2B	63	20	5
e	2001	X	1B	101	3	8
f	2001	Y	C	77	6	4
g	2001	Z	C	101	1	10
h	2001	X	2B	122	14	15
i	2001	Y	1B	91	8	2
k	2001	Z	2B	80	10	6
l	2002	X	C	58	2	11
m	2002	X	1B	70	4	9
n	2002	Z	2B	85	6	13

(b)

name	year	team	hits	HR
a	2000	X	96	11
c	2000	Y	70	19
d			63	20
h	2001	X	122	14
i	2001	Y	91	8
g	2001	Z	101	1
k			80	10
m	2002	X	70	4
n	2002	Z	85	6

Fig. 1 Skyline Cube (SC) and group-by query result. **a** Skyline Cube. **b** Outstanding players

are implemented with Multi-Dimensional DataBase (MDDB) [3]. In MDDB to represent a data cuboid, tuples are partitioned into different cells based on the values of their dimension attributes, where an aggregate function (e.g., SUM) is applied to a measure attribute (e.g., sales) for the tuples partitioned in each cell and the resulting value is assigned to the cell.

Skyline Cube (SC) [4, 5] has been proposed as an extension of Data Cube by using the *skyline* operator [6] to aggregate tuples in a cell instead of the conventional aggregate functions. The *skyline* operator has been received considerable attention in the database and data mining fields. Given a set S of skyline attributes, a tuple t is said to *dominate* another tuple t', denoted by $t \succ_S t'$, if Eq. (1) is satisfied. It is assumed that smaller values are preferable over larger ones. Here, $t[A_i]$ is used to represent the value of the attribute A_i of the tuple t. Given a set D of tuples, Eq. (2) defines the *skyline* operator Ψ on D.

$$(\exists A_i \in S, t[A_i] < t'[A_i]) \wedge (\forall A_j \in S, t[A_j] \leq t'[A_j]) \tag{1}$$

$$\Psi(D, S) = \{t \in D | \nexists t' \in D, t' \succ_S t\} \tag{2}$$

Figure 1a shows an example to explain the SC concept. Each tuple has attributes of *Baseball batters' statistics database*. Regarding *hits*, *HR*(*HomeRun*), and *SB* (*StolenBase*), higher scores are supposed to be preferable over lower ones. If we want to find the outstanding players for each combination of *team* and *year*, a group-by query with dimension attributes $G = \{year, team\}$ and skyline attributes $S = \{hits, HR\}$ can be issued toward the table of Fig. 1a. The table of Fig. 1b is the query result.

In addition to the query type mentioned in the above (See Fig. 2b), range query as another type (See Fig. 2c) can be available for SC, where the *skyline* operator is applied to aggregate tuples satisfying a range condition regarding dimension

(a)

Skyline Cube
$F = \{G_1, G_2, \ldots, S_1, S_2, \ldots, O_1, O_2, \ldots\}$
dimension attribute $G_{pi} \in (G = \{G_1, G_2, \ldots\})$
skyline attribute $S_{qj} \in (S = \{S_1, S_2, \ldots\})$
other attribute $O_{rk} \in (O = \{O_1, O_2, \ldots\})$
skyline preference $m_{qj} \in \{MAX, MIN\}$

(b)

select $G_{p1}, G_{p2}, \ldots,$
$S_{q1}, S_{q2}, \ldots, O_{r1}, O_{r2}, \ldots$
from F
group by $G_{p1}, G_{p2}, \ldots$
skyline of $S_{q1}\ m_1, S_{q2}\ m_2, \ldots$

(c)

select $S_{q1}, S_{q2}, \ldots, O_{r1}, O_{r2}, \ldots$
from F
where $l_1 \leqq G_{p1} \leqq h_1$
and $l_2 \leqq G_{p2} \leqq h_2$
....
skyline of $S_{q1}\ m_1, S_{q2}\ m_2, \ldots$

Fig. 2 Skyline Cube (SC) and SQL statements toward SC. **a** Schema of Skyline Cube. **b** Group-by query. **c** Range query

attributes.[1] Although a range query for SC provides users with flexible querying function, it brings much burdens upon a query processing system. To overcome the difficulty, the work [7] has proposed *SkyCube*-tree, *R*-tree [8] like hierarchical index structure, to implement an efficient range query processing system. Apart from range queries, this paper demonstrates that *SkyCube*-tree can be made use of efficient group-by query processing, though it is originally designed to realize efficient range query processing.

The rest of this paper is organized as follows. Section 2 presents the *SkyCube*-tree. Section 3 discusses group-by query processing methods based on *SkyCube*-tree. Section 4 experimentally evaluates group-by query processing methods. Section 5 mentions the related work. Finally, Sect. 6 concludes the paper.

2 *SkyCube*-Tree

SkyCube-tree is organized in a *R*-tree like hierarchical structure to realize efficient range query processing for SC. In *SkyCube*-tree, a cell of all dimension attributes is treated as a point in a multi-dimensional space. *Skyline* operation result over tuples belonging to a cell is associated to it. In order to process a range query, points located inside the rectangle specified by a range condition are searched first, then *skyline* operation results associated to the found points are used to compute the query result. For a dimension attribute unreferenced in a query, a range is set between $-\infty$ and $+\infty$.

In order to design *SkyCube*-tree, it is required to consider that the *skyline* operator is *holistic* in nature. That is to say, Eq. (3) holds on a set D of tuples regarding a set S of skyline attributes. In other words, the skyline set regarding a subset $S'(\subseteq S)$ cannot be necessarily derived from the skyline set regarding S. *SkyCube*-tree makes use of the *extended skyline* operator [9] to cope with the problem. The *extended skyline* operator is based on *strongly dominance* relation defined in Eq. (4), which strengthens *dominance* relation defined in Eq. (1). According to Eq. (4), a tuple t cannot *dominate* another tuple t' unless all the skyline attribute values of the former

[1]It is possible to impose a range condition on a categorical dimension attribute by converting its domain into a numerical one.

are preferable over those of the latter. t is said to *strongly dominate* t' (denoted by $t \succ_S^+ t'$), if t and t' satisfy Eq. (4). Based on *strongly dominance* relation, Eq. (5) defines the *extended skyline* operator Ψ^+ on D regarding S. Equation (6) holds on D regarding S. By using Ψ^+, the skyline set regarding a subset $S'(\subseteq S)$ is derivable from the skyline set regarding S.

$$\Psi(D, S') \supseteq \Psi(\Psi(D, S), S') \; \textit{for} \; S' \subseteq S \tag{3}$$

$$(\forall A_i \in S, \; t[A_i] < t'[A_i]) \tag{4}$$

$$\Psi^+(D, S) = \{t \in D \mid \nexists \, t' \in D, \; t' \succ_S^+ t\} \tag{5}$$

$$\Psi(D, S') = \Psi(\Psi^+(D, S), S') \; \textit{for} \; S' \subseteq S \tag{6}$$

Figure 3 is the storage structure of *SkyCube*-tree. An *extended skyline* set is variable-length and would occupy a large space region. Therefore, an *extended skyline* set is stored in *extended skyline set space*, which is separated from the body of a *SkyCube*-tree and organized as a sequential file. An *extended skyline* set is pointed from *SkyCube*-tree with its address.[2] A leaf node of a *SkyCube*-tree stores a list of point records, each of which consists of *point coordinates* and *extended skyline set* address. It is noted that a leaf node links to its next leaf node via *nextLink*. An internal node of *SkyCube*-tree stores a list of Minimum Bounding Rectangle (MBR) records, each of which consists of *child node address*, *MBR coordinates*, and *extended skyline set* address. MBR is a minimum rectangular region containing a set of points (or MBRs) which its child node stores and is specified by a pair of its left-bottom point coordinates and its right-upper point coordinates.

R-tree, whose structure is similar to *SkyCube*-tree, obtains an efficient retrieval performance, by balancing its height on insertion/deletion of data. As a result, *R*-tree is not necessarily superior in space efficiency because of an increasing number of nodes. Since a query covering a huge area of SC can be easily expressed, it is inevitably required to reduce the number of I/O operations for accessing to *SkyCube*-tree. Therefore, *SkyCube*-tree should be made compact to reduce the number of I/O operations. While a sizable quantity of data are added to DWH at a certain interval, dynamic insertion/deletion of data is not generally performed until a next interval. From this point of view, *SkyCube*-tree is constructed in a bottom-up way as follows.

(1) A set of tuples is arranged in the order following a space-filling curve[3] for a multi-dimensional space based on all the dimension attributes.
(2) Following the curve, tuples with each combination of dimension attribute values stand in line and can be put together. For these tuples, *extended skyline* set is computed[4] and stored in *extended skyline set space*. Then, a point record is

[2]The size of address data is 8 bytes.

[3]*Z*-curve [10] is employed for the work.

[4] Block Nested Loop (BNL) algorithm [11] is employed for the work.

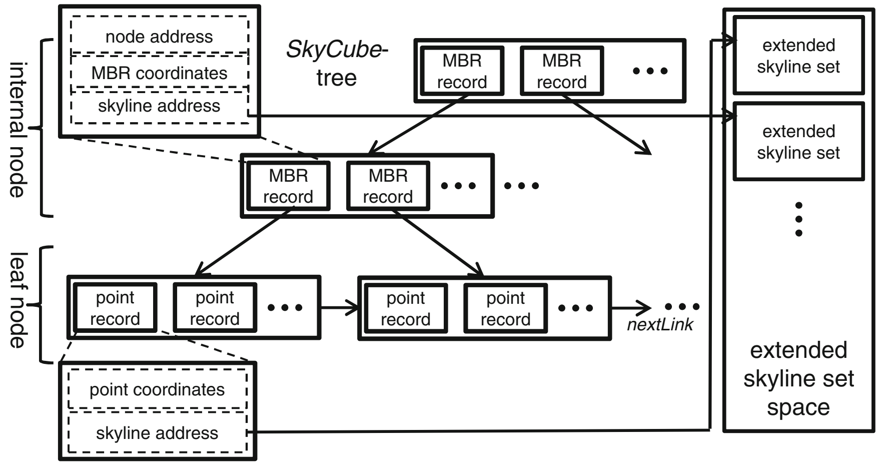


Fig. 3 Storage structure of *SkyCube*-tree

created and inserted into the current leaf node. If an enough space is not left in the node for storing the point record, a new leaf one is created and made current. The new one is referenced from the previous one via *nextLink*.

(3) Following the list of nodes created, MBR is calculated for all the points (or MBRs) stored in each node. Also, *extended skyline* set is computed based on all the *extended skyline* sets stored in the node and stored in *extended skyline set space*. Then, a MBR record is created and inserted into the current internal node. If an enough space is not left in the node for storing the MBR record, a new internal one is created and made current.

(4) If the number of nodes created is one, this unique one is made the root of the *SkyCube*-tree and construction process is finished. Otherwise, **(3)** is repeated.

3 Group-By Query Processing

Regarding a non-empty subset of dimension attributes, a non-empty subset of skyline attributes, and an arbitrary granule of cells, the group-by query result can be computed by accessing to *SkyCube*-tree. Figure 4 shows the flow chart of group-by query processing. All the leaf nodes of *SkyCube*-tree can be traversed rightward via *nextLink* from the leftmost one, which can be accessed from the root node. Each *skyline address* of a point record stored in a leaf node is put together into a corresponding cell. As soon as all the *skyline addresses* of a cell are collected, the *skyline* set of the cell is computed based on the *skyline addresses*, each of which points an *extended skyline* set in the *extended skyline set space*.

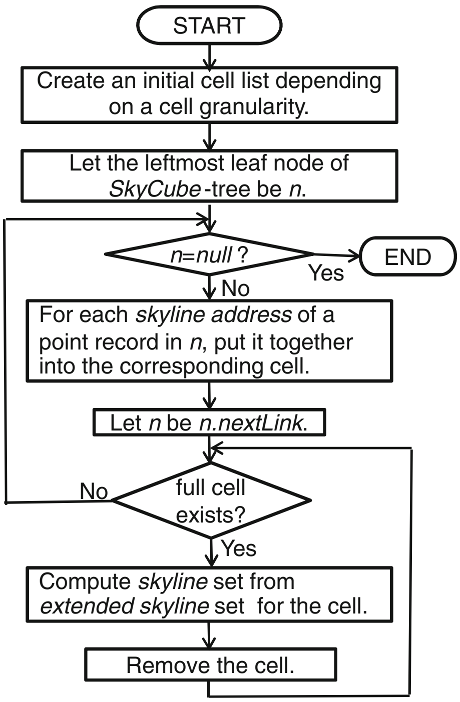

START
Create an initial cell list depending on a cell granularity.
Let the leftmost leaf node of *SkyCube*-tree be *n*.
n=null ?
Yes
END
No
For each *skyline address* of a point record in *n*, put it together into the corresponding cell.
Let *n* be *n.nextLink*.
full cell exists?
No
Yes
Compute *skyline* set from *extended skyline* set for the cell.
Remove the cell.

Fig. 4 Flow chart of group-by query processing

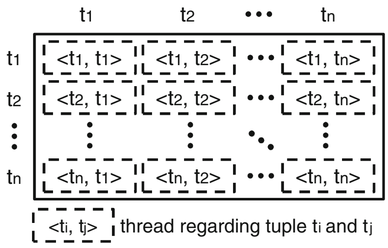


Fig. 5 Threads for parallel skyline computation

Like range query processing for SC [7], parallel processing scheme can be applied to the process "Compute *skyline* set from *extended skyline* set for the cell" shown in Fig. 4. To this end, GPGPU (General-Purpose computing on Graphics Processing Units) [12] has been used to implement parallel processing, where massive light-weight threads can be dealt with comparatively easily.

Let U be *extended skyline* set. For tuple $t(\in U)$, whether t belongs to a *skyline* set or not can be judged by checking if $t'(\in U - \{t\})$ *dominates* t (See Eq. (2)). Furthermore, *dominance* relation over t and t' can be checked independently from other combinations. Based on the property, parallel processing can be executed for the process "Compute *skyline* set from *extended skyline* set for the cell". Figure 5 shows a set of threads to be executed in parallel with GPU. For each combination of $t_i, t_j(\in U)$, a thread is dedicated to checking *dominance* relation over t_i and t_j. If $t_j \succ_S t_i$ holds, t_i is removed from the final *skyline* set. Although *dominance* relation over t_k and $t_k (k = 1, \dots, n)$ is self-evident, a thread is also generated for simplicity.

4 Experiments

In this section, *SkyCube*-tree and group-by query processing methods are experimentally evaluated. First, the settings of experiments are described. Then, the size of *SkyCube*-trees is presented. Finally, the processing time for group-by queries is presented.

4.1 Experimental Settings

The experiments are conducted on an Intel Core i5-3550 3.3 GHz PC with 8 GB memory and a NVIDIA Quadro K4000 with 768cores and 3 GB memory. C language is used to implement programs. Also, CUDA [12], designed by NVIDIA, is employed as GPGPU for working with C.

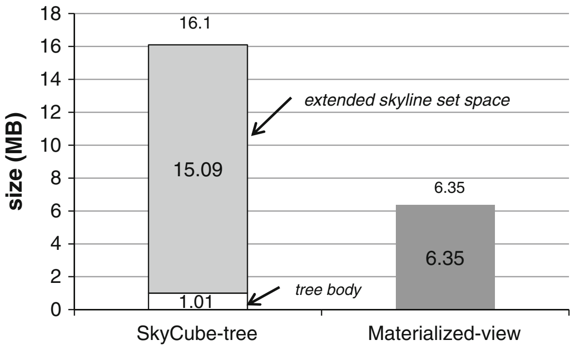
A synthetic dataset is generated to carry out the experiments. It contains 100,000 tuples, with a total of 14 attributes $(a_1, a_2, \dots, a_{14})$: 5 attributes $a_1, \dots, a_5$ are for dimension, and 9 attributes $a_6, \dots, a_{14}$ are for skyline. The domain size of each dimension attribute is 10, where attribute values are generated from the uniform distribution. On the other hand, the domain size of each skyline attribute is 1,000. Specifically, a_6, a_7, a_8 are independently generated from Gaussian distribution, a_9, a_{10}, a_{11} are correlated with a_6, and a_{12}, a_{13}, a_{14} are anti-correlated with a_6.

In the experiments, *SkyCube*-tree and the proposed query processing methods are compared with the materialized view based one. A materialized view is a cuboid which is precomputed and stored for answering queries. When a query is issued, it can be processed by using a materialized view whose attribute set covers dimension attributes and skyline attributes referenced in the query. For the experimental comparison, a materialized view organized in a sequential file is constructed. However, it is noted that the materialized view is different from that of the work [4, 5], because the former maintains *extended skyline* sets.

4.2 Storage Size of Skyline Cube

Figure 6 shows the storage size of the *SkyCube*-tree and the materialized view, each of which is constructed for the synthetic dataset. In the experiments, the page size is set to 4KB and the number of nodes in *SkyCube*-tree is 253. The storage structure of *SkyCube*-tree consists of *tree body* and *extended skyline set space*. The size of the

Fig. 6 Storage size of *SkyCube*-tree and materialized view



SkyCube-tree is 16.1 MB, whose *tree body* occupies 1.01MB and whose *extended skyline set space* occupies 15.09MB. Meanwhile, the size of the materialized view is 6.35MB. Given a dimension attribute set G and a skyline attribute set S, the number of possible cuboids regarding a non-empty subset of G and a non-empty subset of S is $(2^{|G|} - 1)\ (2^{|S|} - 1)$. The materialized view is constructed regarding all the dimension attributes and all the skyline attributes, which corresponds to the largest cuboid among all the possible ones.

The total size of the *SkyCube*-tree is 2.5 times as large as that of the materialized view. Although a unique *SkyCube*-tree is enough to make answers of all the queries, the number of materialized views to be potentially needed is $(2^{|G|} - 1)\ (2^{|S|} - 1)$, as is mentioned in the above. Therefore, the storage size of *SkyCube*-trees is sufficiently allowable. The main difference of storage size between the *SkyCube*-tree and the materialized view constructed resides in the back portion of *extended skyline set space*, where a family of *extended skyline set* referenced from internal nodes of the *SkyCube*-tree are stored. These *extended skyline sets* are prepared for efficiently processing a range query whose size is that of MBR corresponding to an internal node or more. It is noted that the portion is not used for group-by query processing and extra I/O times are not needed consequently.

4.3 Group-By Query Processing Time

Regarding group-by query processing time, *SkyCube* based sequential processing (SG-) method, *SkyCube* based parallel processing (PG-) method, and materialized view based sequential processing (MG-) method are compared. Given the number of dimension attributes (*dimension#*), the number of skyline attributes (*skyline#*), and *cell-granularity*, 100 queries are generated and query processing times are averaged. For each query, a subset of dimension attributes and a subset of skyline attributes are randomly chosen. Each dimensional size of a cell is set to the nth root of *cell-granularity*, if *dimension#* is n.

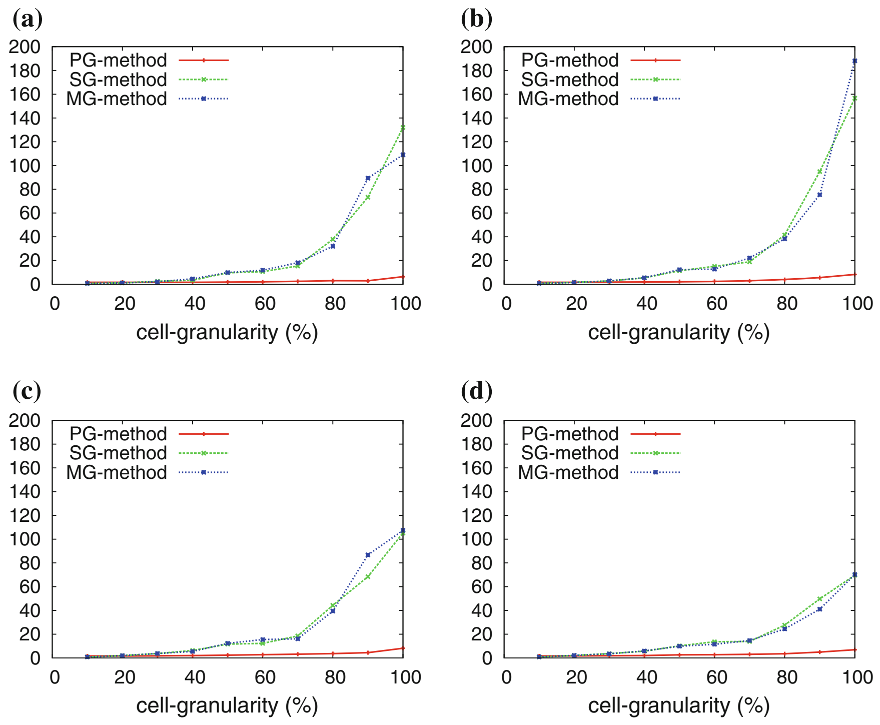


Fig. 7 Group-by query processing time for synthetic dataset (seconds). **a** *dimension*# = 3, *skyline*# = 2. **b** *dimension*# = 3, *skyline*# = 3. **c** *dimension*# = 3, *skyline*# = 4. **d** *dimension*# = 3, *skyline*# = 5

For MG-method, the materialized view is firstly constructed regarding all the dimension attributes and all the skyline attributes. To make the experiments fair, additional materialized views are constructed regarding 3 or more dimension attributes and 2 or more skyline attributes, until the total size of materialized views becomes not less than that of the corresponding *SkyCube*-tree. The number and each dimension (skyline) attribute are randomly chosen. MG-method processes a query by using the materialized view whose attributes set minimally covers dimension attributes and skyline attributes referenced in the query.

Figure 7 shows the experimental results where *dimension*# is set at 3, *skyline*# is varied from 2 to 5, and *cell-granularity* is varied from 10 to 100 % with increments of 10 %. The tendency of each graph is roughly the same as follows:

- The processing time with SG-method is nearly equal to that with MG-method, though *SkyCube*-tree is originally designed for efficient range query processing.
- The graphs rise to the right. The larger *cell-granularity* becomes, the more processing time is needed.

- The processing time with PG-method[5] is comparatively small and stable. It is around 10 % of that with MG-method, even if *cell-granularity* is over 80 %.

Generally speaking of the *skyline* operator, if a set of skyline attributes includes one attribute and another anti-correlated attribute both, the size of the resultant skyline set increases rapidly and its computational cost increases as a consequence. In case of *skyline#* between 2 and 5 for the synthetic dataset, occurrence probability that a set of skyline attributes includes one attribute and another anti-correlated attribute both is 0.30556, 0.64286, 0.84127, and 0.94444 respectively.

Since a group-by query includes the entire dataset as its processing range, each method is required to access to all the *extended skyline* sets. While MG-method does it by sequentially scanning a materialized view, both SG-method and PG-method do it similarly with a *SkyCube*-tree, by firstly going down to the leftmost leaf node from the root and then traversing leaf nodes from left to right via *nextLink*. This is the reason why both SG-method and MG-method take almost the same amount of time.

5 Related Work

Skycubes [13] is an efficient evaluation algorithm for the *skyline* operator. Let a skyline attribute set be S. Similarly to materialized views for a data cube, *skycubes* precomputes *skyline* operation result regarding a non-empty subset of *S* and organizes all the operation results of distinct skyline attribute subsets in a lattice structure. When a skyline operation is requested, the corresponding query result in the lattice is returned as an answer. However, *skycubes* is simply dedicated to evaluating the *skyline* operator. Meanwhile, queries for SC must take an additional set of dimension attributes into consideration.

Range-Sum (*Range-Max*) [14] is an algorithm for processing range queries which use function SUM (MAX) to aggregate tuples belonging to each cell of a data cube. To perform query processing efficiently, the algorithms rely on respective auxiliary index structures *prefix-Sum* and *b-ary* tree. Both index structures partition each dimension attribute domain by an even interval, which leads to efficient searching of index regions relating to query processing. However, they might be inferior in space efficiency, if value distribution of a dimension domain is to some extent uneven and/or sparse.

Ag+-tree [15, 16] is a hierarchical index structure dedicated to processing range queries regarding a data cube. Similarly to *SkyCube*-tree, a unique Ag+-tree is constructed for a data cube. Let the number of dimension attributes be *n*. A node of Ag+-tree consists of *n* pages, each of which is dedicated to a corresponding attribute, and a single page possessing several kinds of aggregation values, where page is a unit of I/O operations. However, it lacks high space efficiency, while *SkyCube*-tree is compactly constructed in a bottom-up way.

[5] Note that it includes the time to transfer data between main memory of CPU and RAM memory of GPU.

As mentioned before, the work [4, 5] proposed SC as an extension of Data Cube. While materialized views are used to process group-by queries in the work, *SkyCube*-trees are used to do that in our work. Additionally, another difference resides in that range queries can be processed by using the same *SkyCube*-trees in our work [7].

6 Conclusion

Since a group-by query for SC includes the entire dataset as its processing range, the query processing time is potentially large. Regardless of the difficulty, this paper demonstrates that group-by queries for SC are processed efficiently with *SkyCube*-tree, though it is originally designed to realize efficient range query processing. While its storage size is sufficiently allowable, the processing time with SG-method is nearly equal to that with MG-method and the processing time with PG-method is comparatively small and stable. Even though *cell-granularity* is over 80 %, the processing time with PG-method is around 10 % of that with MG-method.

A hierarchy consisting of several levels of a dimension attribute is supposed to implement traditional OLAP tools. For example, {"the first half of the year", "the second half of the year"} might be one level, {"the first quarter of the year", "the second quarter of the year", …} might be another level, and the others so on for a fiscal period attribute. *Drill-up* (*Drill-down*) operator ascends (descends) a hierarchy, which makes group-by queries executed under *cell-granularity* corresponding to some level of a hierarchy. However, it is noted that *SkyCube*-tree based query processing methods are more flexible, since they can make group-by queries executed under any of *cell-granularity*.

Our future work is related to reduction of a *skyline* set which is potentially huge for a large number of skyline attributes. To this end, two kinds of operators might be available. One is the *k-dominant skyline* operator [17] and the other is the *top-k* query [18].

References

1. Codd, E.F.: Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate, Technical report. E.F.Codd and Associates (1993)
2. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: A relational aggregation operator generalizing group-by, cross-tabs and subtotals. In: Proceedings of the 12th Int'l Conference on Data Engineering, pp. 152–159 (1996)
3. Agrawal, R., Gupta, A., Sarawagi, S.: Modeling multidimensional databases, In: Proceedings of the 13th Int'l Conference on Data Engineering (1997)
4. Yiu, M.L., Lo, E., Yung, D.: Measuring the sky: on computing data cubes via skylining the measure. IEEE Trans. Knowl. Data Eng. **24**(3), 492–505 (2012)
5. Luk, M.N., Yiu, M.L., Megiddo, N., Lo, E.: Group-by skyline query processing in relational engines. In: Proceedingsc of the 18th ACM Conference on Information and Knowledge Management, pp. 1433–1436 (2009)

6. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th Int'l Conference on Data Engineering, pp. 421–430 (2001)
7. Sato, H., Usami, T.: Range query processing in OLAP skyline cubes. IEEJ Trans. Electron. Inf. Syst. **136**(4) (2016). (in Japanese, to be published)
8. Guttman, A.: R-Trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD Int'l Conference on Management of Data, pp. 45–57 (1984)
9. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: SKYPEER: efficient subspace skyline computation over distributed data. In: Proceedings of the 23rd Int'l Conference on Data Engineering, pp. 416–425 (2007)
10. Orenstein, J., Merrett, T.H.: A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Data Base Systems, pp. 181–190 (1984)
11. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
12. Sanders, J., Kandrot, E.: CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional (2010)
13. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings the 2005 Int'l Conference on Very Large Data Bases, pp. 241–252 (2005)
14. Ho, C.T., Agrawal, R., Megiddo, N., Srikant, R.: Range queries in OLAP data cubes. In: Proceedings of the 1997 ACM SIGMOD Int'l Conference on Management of Data, pp. 73–88 (1997)
15. Feng, Y., Makinouchi, A.: Indexing for range-aggregation queries on large relational datasets. Int'l J. Database Theory Appl. **3**(4), 1–14 (2010)
16. Feng, Y., Makinouchi, A.: Ag+-tree: an index structure for range-aggregation queries in data warehouse environments. Int'l J. Database Theory Appl. **4**(2), 51–64 (2011)
17. Chan, C-Y., Jagadish, H.V., Tan, K-L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of the 2006 ACM SIGMOD Int'l Conference on Management of Data, pp. 503–514 (2006)
18. IIyas, H.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4) (2008). Article 11