Efficient Processing of Ranked Queries with Sweeping Selection*

Wen Jin[1], Martin Ester[1], and Jiawei Han[2]

[1] School of Computing Science,Simon Fraser University
{wjin, ester}@cs.sfu.ca
[2] Department of Computer Science, Univ. of Illinois at Urbana-Champaign
hanj@cs.uiuc.edu

Abstract. Existing methods for top-k ranked query employ techniques including sorting, updating thresholds and materializing views. In this paper, we propose two novel index-based techniques for top-k ranked query: (1) indexing the layered skyline, and (2) indexing microclusters of objects into a grid structure. We also develop efficient algorithms for ranked query by locating the answer points during the sweeping of the line/hyperplane of the score function over the indexed objects. Both methods can be easily plugged into typical multi-dimensional database indexes. The comprehensive experiments not only demonstrate that our methods outperform the existing ones, but also illustrate that the application of data mining technique (microclustering) is a useful and effective solution for database query processing.

1 Introduction

Rank-aware query processing is important in database systems. The answer to a top-k query returns k tuples ordered according to a specific score function that combines the values from participating attributes. The combined score function is usually linear and monotone with regard to the individual input. For example, given a hotel database with attributes of x (*distance* to the beach) and y (*price*), and the score function $f(x, y) = 0.3x + 0.7y$, the top-3 hotels are the best three hotels that minimize f.

The straightforward method to answer top-k queries is to first calculate the score of each tuple, and then output the top-k tuples from them. This *fully-ranked* approach is undesirable for querying a relatively small number k of a large number of objects. Several methods towards improving the efficiency of such queries have been developed, but they are either specific to joined relations of two dimensions [16], or incompatible with other database indexing techniques [4, 8], or computationally expensive [11]. In this paper, we propose two novel index-based techniques for top-k ranked query. The first method is *indexing the layered skyline based on the skyline operator* [2], whereas the second, motivated by a major data mining technique-microclustering, is *indexing microclusters of the dataset into a grid structure*. We develop efficient algorithms for answering the ranked query by locating the answer points during sweeping the line/hyperplane of the score function over the indexed objects. Both methods can easily be plugged into typical multi-dimensional database indexes. For example, the

* The work was supported in part by Canada NSERC and U.S. NSF IIS-02-09199.

layered skyline can be maintained in a multi-dimensional index structure with blocks described by (1) MBR (Minimum Bounded Rectangle) such as R-tree [9] and R*-tree [3], or (2) spherical Microcluster such as CF-tree [19] and SS-tree [18]. The comprehensive experiments not only demonstrate the high performance of our methods over the existing ones, but also illustrate that the microclustering technique is an effective solution to top-k query processing.

The rest of the paper is organized as follows. Section 2 presents the foundations of this paper. Section 3 and Section 4 give KNN-based and Grid-based sweeping algorithms for the ranked queries respectively and illustrate their plug-in adaptations for R-trees and CF-trees. Section 5 reviews related work. We present our experimental results in Section 6 and conclude the paper in Section 7.

2 Foundations

Let a d-dimensional dataset be X, and a linear monotone score function be $f(x) = \sum_{i=1}^{d} a_i \cdot x_i$ where $x \in X$, and a_i is the weight on the attribute value x_i of x such that $\sum_{i=1}^{d} a_i = 1$, $0 \leq a_i \leq 1$. Without loss of generality, we assume the lower the score value, the higher the rank of the object. A top-k ranked query returns a collection $T \subset X$ of k objects ordered by f, such that for all $t \in T$, $x \in X$ and $x \notin T$, $f(t) \leq f(x)$. We say $p = (p_1, \ldots, p_d) \in X$ *dominates* another object $q = (q_1, \ldots, q_d) \in X$, denoted as $p \succ q$, if $p_i \leq q_i$ $(1 \leq i \leq d)$ and at least there is one attribute, say, the jth attribute $(1 \leq j \leq d)$, $p_j < q_j$. Hence, q is a *dominated object*. The *skyline* operator [2] is defined as objects $\{s_1, s_2, \ldots, s_m\} \subset X$ that are not dominated by any other object in X. A *multilayered skyline* [6], which is regarded as a stratification of the dominating relationship in the dataset, is organized as follows: (1) the first layer skyline L_1 is the skyline of X, and (2) the ith layer $(i > 1)$ skyline L_i is the set of skyline objects in $X - \bigcup_{j=1}^{i-1} L_j$. As for two objects $p \in X, q \in X$, if $p \succ q$ then $f(p_1, \ldots, p_d) < f(q_1, \ldots, q_d)$, so we can derive the following interesting properties.

Lemma 1. *Given X and f, for any $i < j$, if object $q \in L_j$, there exists at least one object $p \in L_i$ s.t. $f(p_1, \ldots, p_d) < f(q_1, \ldots, q_d)$.*

Theorem 1. *Given $K(K \geq k)$ layers of skyline $L_1, \ldots, L_K$, any top-k tuples w.r.t. f must be contained in $\bigcup_{j=1}^{K} L_j$, i.e. in the first K skyline layers.*

Definition 1. (MicroCluster[19]) *The MicroCluster C for n objects is represented as $(n, \overline{CF1(C)}, \overline{CF2(C)}, \overline{CF3(C)}, r)$, where the linear sum and the sum of the squares of the data values are maintained in $\overline{CF1(C)}$, $\overline{CF2(C)}$ respectively. The centroid of C is $\overline{CF3(C)} = \frac{\overline{CF1(C)}}{n}$, the radius of C is $r = (\frac{\overline{CF2(C)}}{n} - (\frac{\overline{CF1(C)}}{n})^2)^{\frac{1}{2}}$.*

Therefore, maintaining K layers of skylines is enough to answer any top-k ranked query with $k \leq K$. The layered skyline can be organized into rectangle-like and sphere-like *blocks*, hence supported by two types of multi-dimensional database indexes. We choose R-tree and CF-tree as typical representatives of these two types of indexes. The basic storage of a *block* in R-tree is a MBR in a leaf node, and is a microcluster in leaf node in CF-tree [19]. We compute the $K(k \leq K)$ layers of skyline objects by recursively applying existing skyline computing algorithms such as [14], and only store layered skyline as *blocks* in the leaf nodes of index structures.

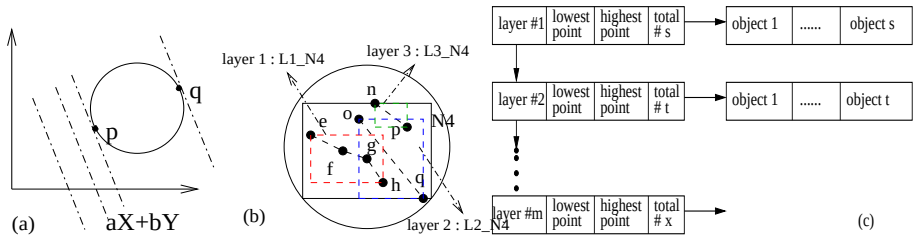

Fig. 1. (a)Contact Points (b) Layered-Skyline in N4 (c) Linked List for Layered Skyline in a Block

3 A KNN-based Sweeping Approach for Top-k Queries

Here we present methods of two levels of a hyperplane sweeping: (1) sweeping over blocks such that the I/O cost of accessing blocks is minimized, and (2) sweeping within a block such that the CPU cost of processing objects within a block is minimized.

(1) Sweeping over blocks. During the sweeping process, the hyperplane always contacts the best point first, the next best point second, and so on. Based on the indexed K skyline layers, we develop an efficient branch-and-bound algorithm for top-k query similar to the optimal KNN search algorithm [15] [16]. Each block has a lowest/highest point corresponding to the lowest/highest score. In the algorithm, a sorted queue Q is used to maintain the processed points and blocks in ascending order of their score. The algorithm starts from the root node and inserts all its blocks to the queue. The score for a data block is the score for its lowest point. The first block in the queue will then be expanded. If the entry is a leaf *block*, we will access its data with some strategy. In the expanding process, we also keep track of how many data points are already present, and if an object or a block is dominated by enough ($> k$) objects (in some blocks) lining in the queue before it, then it can be pruned. The expanding process stops when the queue has k objects in the front.

Algorithm 1 Branch-and-Bound Ranking (BBR) Method.
Input: k, and a multi-dimensional index tree
Output: Top-k answer in Q
Method:

1. Q := Root Block;
2. WHILE top-k tuples not found DO
3. F := the first non-object element from Q;
4. S := SweepIntoBlock(F); //S is a set of blocks and/or objects
5. FOR each block/object s in S Do
6. IF more than k objects in Q having smaller score than s
7. Discard s;
8. ELSE Insert s to Q;
9. Output k objects from Q;

Algorithm 1 have different implementations for **SweepIntoBlock**. It can simply expand the block and access all the belonged objects (noted as **BBR1**). For a MBR in R-tree, the lowest/highest points are the lower/upper right corner points, while for a microcluster in CF-tree, these are two contact points of the sweeping hyperplane to

the sphere, shown as p and q in Fig. 1(a). Given a sweeping hyperplane $y = \sum_{i=1}^{d} a_i \cdot x_i$ and a microcluster F with radius R of the origin: $F(x_1, \ldots, x_d) = \sum_{i=1}^{d} {x_i}^2 - R^2 = 0$ (1). To obtain p and q, we only need solve $\nabla F(x'_1, \ldots, x'_d) = c \cdot \boldsymbol{A}$ (2) together with (1). Here c is a free variable, $\boldsymbol{A} = a_1, \ldots, a_d$, and $\nabla F(x_1, \ldots, x_d)$ which works as the gradient of F at X$(x'_1, \ldots, x'_d)$, is $(\frac{\partial F}{\partial x_1}(x'_1, \ldots, x'_d), \ \ldots \ , \ \frac{\partial F}{\partial x_d}(x'_1, \ldots, x'_d) \ )$.

(2) Sweeping within layered blocks. In order to avoid processing unnecessary objects in each block, we make use of the layered skylines since they give a contour of the data distribution, and develop an efficient sweeping within-layered-blocks method (noted as **BBR2**), as a procedure SweepIntoBlock in Algorithm 1. Suppose the *block* in a leaf node has m layers of skylines, the lowest/highest object as well as the total number of objects for that layer is maintained. As shown the node in Fig. 1(b), objects e, f, g and h are layer-1 skyline objects in node N_4, we put all skyline objects in layer 1 minimally hyperectangle-bounded by a pseudo-node denoted as $L_1\_N_4$. The linked list storage structure for the layered skylines is shown in Fig. 1(c), where the header is the summarization information of the pseudo-node which links to its bounding skyline objects. Now if a leaf *block* is chosen from the queue, we only expand the pseudo-node that has the best lowest point according to the score function.

4 A Grid-based Sweeping Approach for Top-k Queries

Although the KNN-based approach can efficiently obtain the top-k objects, it may still visit and compare all the objects in a block even when the layering technique is applied. In this section, we present an alternative grid-based method for more efficiently organizing the objects. Since the user-specified weights of a score function will often have a fuzzy rather than a crisp semantic, approximate, query processing seems to be acceptable if this allows significantly improved response time. The basic idea is to build a grid-like partition of the *blocks* and access the objects within a block along the grid. For CF-tree, a shell-grid partition is made (the R-tree case can be adapted by bounding a MBR over any microcluster). The block entries are then assigned to the grid cells, and the sweeping algorithm is applied. All objects of the current grid cell are accessed before those of other grid cells. This approach reduces the number of comparisons, but it may lead to a non-exact result if k answer objects are found before a further grid cell with better objects is accessed. The overview of CF-tree with the shell grid partition is shown in Fig. 2, where 2(b) depicts the anatomy of an intermediate microcluster node D in 2(a). A single **shell grid cell** (or **cell**) is shown in 2(c). We can enforce the radius of the leaf microcluster to be smaller than ε when building the CF-tree [13]. Motivated by the Pyramid indexing [1], we propose a novel idea of building shell grid partitions for microcluster nodes. The partition made is in partial shells and the center of the sphere is pre-computed. Assume the center is

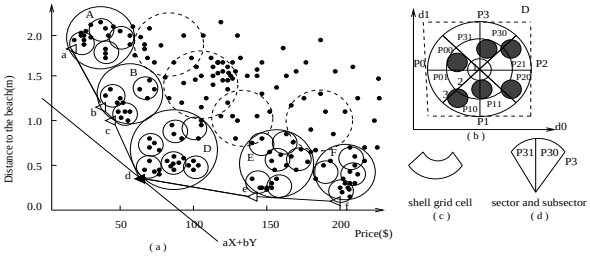

Fig. 2. Shell-Grid Partition of Microclusters

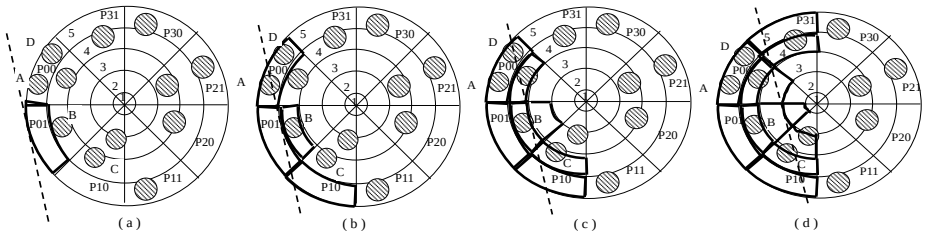

Fig. 3. Sweeping a Shell Grid

$o = (o_0, \ldots, o_{d-1})$. We split the spherical space into $2^{d-1} \cdot 2d$ fan-shaped partitions, each having the center as the top, and $1/(2^{d-1} \cdot 2d)$ of the $(d-1)$-dimensional spherical surface as the base. The sphere is split into $2d$ **sectors** $P_0, \ldots, P_{2d-1}$ according to the square cube with $2d$ surfaces (dashed in Fig. 2(b) enclosing the sphere is 2-dimension case), as P_3 in Fig. 2(d). Using hyperplane perpendicular to each axis and passing through the sphere center to split the whole space, each sector P_i is divided into 2^{d-1} **subsectors** $P_{i0}, \ldots, P_{i(2^{d-1}-1)}$, as P_{31} and P_{30} in Fig. 2(d). Then the whole space is divided with parallel spherical **shells** starting from the center. We have the following property:

Lemma 2. *For any object x in sector P_i where $i < d$, it satisfies: for any dimension j, $0 \leq j < d$, $i \neq j$, $|o_i - x_i| \leq |o_j - x_j|$; for sector P_i where $i \geq d$, it satisfies: for any dimension j, $0 \leq j < d$, $j \neq (i-d)$, $|o_{i-d} - x_{i-d}| \geq |o_j - x_j|$.*

We further number these subsectors from 0 to $2^{d-1} - 1$ in $(d-1)$ binary format $s_0, \ldots, s_{d-1}$. If $i < d$, the bit s_i does not exist in the binary string, otherwise s_{i-d} is excluded. For each subsector, if $s_j = 1$, the belonged points have $x_j > x_i$, and it is opposite for $s_j = 0$. Each subsector has $2^{d-1} - 1$ direct neighbors in the same sector, and another $(d-1) \cdot 2^{d-2}$ in the neighboring sector. As illustrated in Fig. 3, a useful sweeping property is: *the sweeping process explores first the outmost shell grid cell of the sector which the sweeping hyperplane tangent contacts, then goes to its directed neighboring cells in the same shell. If there is no data in those neighboring cells, sweeping should go to the inner cell in the same sector directly.* Generally we can first calculate the two contacting points and the sector number P_{mn} that the sweeping plane contacts first, and then start the hierarchical sweeping process. A sorted queue Q is used to store the expanding entities including microclusters and a pseudo-node that has candidate sector number and shell number information.

Algorithm 2 A Shell-Grid Ranking(SGR) Method.
Input: CF-tree with Grid Shell Partitions, k.
Output: Top-k answers in list T.
Method:

1. Calculate standard contacting points and subsector number P_{mn}; $Q = \emptyset$; $T = \emptyset$;
2. Insert into Q the outmost cell of root node of CF;
3. WHILE the first k tuples are not found DO
4. Remove the first entity E in Q;
5. IF E is a cell

6. insert blocks in subsector P_{mn} and its direct neighbors with pseudo-entities;
7. ELSE IF E is an intermediate node
8. insert into Q the outmost cell of E;
9. ELSE IF E is a pseudo-entity
10. insert into Q blocks in its neighbor subsectors of the same cell with pseudo-entities;
11. ELSE IF E is a leaf
12. add E to T;
13. Output k points from T;

To analyze the error bound measured in the score difference of the objects, we observe the sweeping process in Fig. 4, some objects in other microcluster (i.e., MC2) are better than those in the current selected microcluster (i.e., MC1). In the extreme case, q=$(x'_1, \ldots, x'_d)$ is computed as part of the answer instead of w whose score is slightly larger than that of $p = (x_1, \ldots, x_d)$. $f(q) - f(w) < f(q) - f(p) = a_1 \cdot (x'_1 - x_1) + \cdots + a_d \cdot (x'_d - x_d) < 2 \cdot \varepsilon \cdot \sum a_i = 2 \cdot \varepsilon$, so the maximum error is $O(\varepsilon)$.

5 Experiments

In this section, we report the results of our experiments performed on a Pentium III 800MHz machine of 512M RAM running WindowsXP. We implemented our methods and Onion in C++, and obtain PREFER in www.db.ucsd.edu/prefer. Two types of datasets of 100,000 records with 5 attributes in independent/correlated distribution were generated by the data generator of [2].

(1) Comparison of BBR and SGR. Figs. 5 and 6 show that all the algorithms have better performance for the correlated dataset. BBR2 runs much faster than BBR1 due to its smaller number of visited objects, while SGR ($\varepsilon = 10$) is best and is an order of magnitude faster than BBR2. When ε changes from 5 to 10 and 15 in the independent dataset, runtime decreases due to the decreasing number of visited microclusters, and the error rate as well as coverage rate become relatively higher due to the increasing size of microclusters (Figs. 7, 8 and 9).

(2) Comparison with Onion and PREFER. We compare the time to construct K layers of skylines for BBR, SGR, and K layers of convex hulls for Onion($K = 200$). When the dimension varies from 2 to 5, SGR uses the least time (Fig. 10). BBR2 is

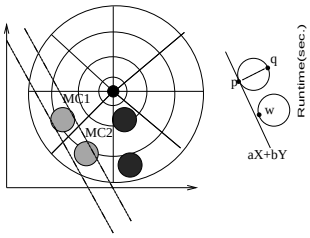

Fig. 4. Error Bound

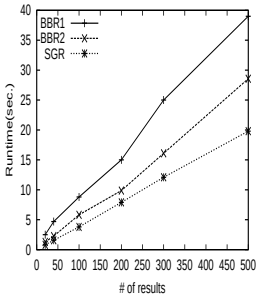

Fig. 5. Query time (1)

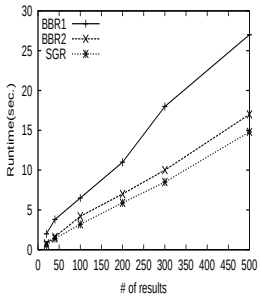

Fig. 6. Query time (2)

much faster than Onion when the dimension increases, as the complexity of computing convex hull increases exponentially with dimensions. When k changes, all perform better on the correlated dataset due to the smaller number of skyline objects. SGR is still the best due to its high efficient sweeping, BBR2 ranks second and PREFER queries faster than Onion (Figs. 11, 12). When k increases, the coverage rate of the answers of SGR is higher than that of PREFER (Figs. 13 and 14). Because correlated dataset has less number of skyline, the size of materialized views for PREFER will be reduced, which leads to the lower coverage rate than the independent dataset.

6 Related Work

The top-k ranked query problem was proposed by Fagin in the context of multimedia database systems [7], and methods can be categorized into three types: **(1) Sorted accessing and ranking** mainly applies some strategies to sequentially search the sorted list of each attributes until the top-k tuples are retrieved. [17] proposed different ways to improve that in [7]. Further, a "threshold algorithm (TA)" [8] is developed to scan all query-relevant index lists in an interleaved manner. **(2) Random accessing and ranking** supports mainly random access over the dataset until the answers have been retrieved. [10] uses foot-rule distance to measure the two rankings and model the rank problem as the minimum cost perfect matching problem, whereas [5] proposes to translate the top-k query into a range query in database. **(3) Pre-materialization and rank indices** organizes the tuples in a special way, then applies similarity match for the answer of ranked query. In [4], an index is built on layered convex hulls for all points. Such index for large databases is expensive due to the convex hull finding complexity. [11, 12] propose to pre-materialize views of different top-k query answers. When the query's score function is close to a view's, a small number of the tuples in that view is necessary for the top tuples, and the query result can be produced in a pipelined fashion. As there is no guarantee how many tuples should each view store for query, it often stores the whole dataset in each view. [16] proposes a ranked index to support top-k query($k \leq K$) but it only applies to two dimensions and can have a huge number of materialized partitions.

7 Conclusions

Rank-aware query processing has recently emerged as an important paradigm in database systems, and only few existing methods exploit materialization and index

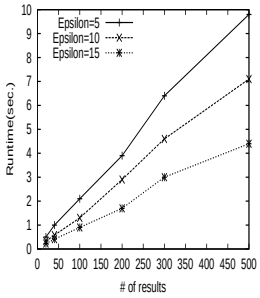

Fig. 7. Effect of Eps(1)

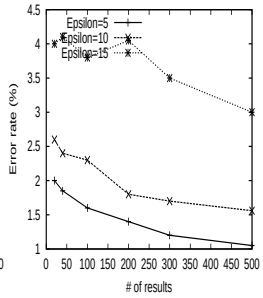

Fig. 8. Effect of Eps(2)

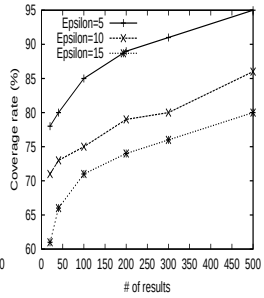

Fig. 9. Effect of Eps(3)

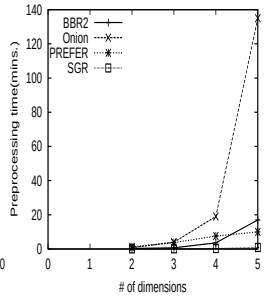

Fig. 10. Preprocessing

structures. In this paper, we propose indexing layered skyline and shell-grid microclusters for top-k ranked query, and present methods sweeping the hyperplane of the score function over the indexed objects. Our methods can be easily adapted to existing multi-dimensional index structures. The experimental results demonstrate the strength of our methods and the usefulness of the microclustering technique in top-k query processing.

References

1. S. Berchtold, C. Bhm and H. P. Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. *SIGMOD* 1998.
2. S. Borzsonyi, D. Kossmann, and K. Stocker. The Skyline Operator. *ICDE* 2001.
3. N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles *SIGMOD* 1990.
4. Y. C. Chang, L. D. Bergman, V. Castelli, C. S. Li, M. L. Lo and J. R. Smith. Onion Technique:Indexing for Linear Optimization Queries. *SIGMOD* 2000.
5. S. Chaudhuri and L. Gravano. Evaluating Top-k Selection Queries. *VLDB* 1999.
6. T. Cormen, C. E. Leiserson, et al. Introduction to Algorithms, The MIT Press, 2001.
7. R. Fagin. Fuzzy Queries in Multimedia Database Systems. *PODS* 1998.
8. R. Fagin. et al. Optimal Aggregation Algorithms for Middleware. *PODS* 2001.
9. A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD* 1984.
10. S. Guha, et al. Merging the Results of Approximate Match Operations. *VLDB* 2004.
11. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. *SIGMOD* 2001.
12. V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.* 2004.
13. Wen Jin, Jiawei Han, Martin Ester Mining Thick Skylines over Large Databases. *PKDD* 2004.
14. D. Papadias, Y. F. Tao, G. Fu and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. *SIGMOD* 2003.
15. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. *SIGMOD* 1995.
16. P. Tsaparas, et al. Ranked Join Indices. *ICDE* 2003.
17. E. L. Wimmers, L. M. Haas, M. T. Roth and C. Braendli. Using Fagin's Algorithm for Merging Ranked Results in Multimedia Middleware. *CoopIS* 1999.
18. D. A. White and R. Jain. Similarity Indexiug with the SS-tree. *ICDE* 1996.
19. T. Zhang, R. Ramakrishnan, and M. Livny BIRCH: an efficient data clustering method for large databases. *SIGMOD* 1996.

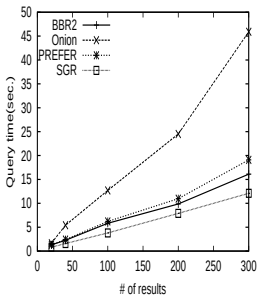

Fig. 11. Query(1)

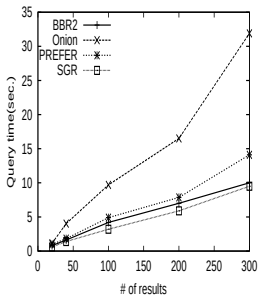

Fig. 12. Query(2)

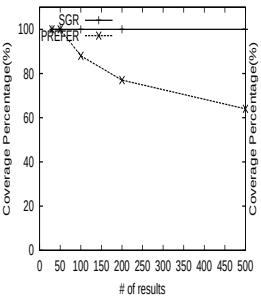

Fig. 13. Quality(1)

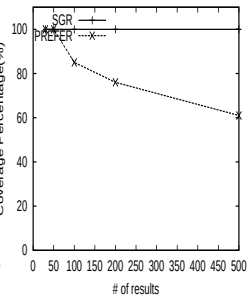

Fig. 14. Quality(2)