Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK 2000),
Greenwich, U.K., 2000.

The Pruning Power: Theory and Heuristics for Mining Databases with Multiple *k*-Nearest-Neighbor Queries

Christian Böhm, Bernhard Braunmüller, Hans-Peter Kriegel

University of Munich, Oettingenstr. 67, D-80538 Munich, Germany
{boehm,braunmue,kriegel}@informatik.uni-muenchen.de

Abstract. Numerous data mining algorithms rely heavily on similarity queries. Although many or even all of the performed queries do not depend on each other, the algorithms process them in a sequential way. Recently, a novel technique for efficiently processing *multiple similarity queries* issued simultaneously has been introduced. It was shown that multiple similarity queries substantially speed-up query intensive data mining applications. For the important case of multiple *k*-nearest neighbor queries on top of a multidimensional index structure the problem of scheduling directory pages and data pages arises. This aspect has not been addressed so far. In this paper, we derive the theoretic foundation of this scheduling problem. Additionally, we propose several scheduling algorithms based on our theoretical results. In our experimental evaluation, we show that considering the maximum priority of pages clearly outperforms other scheduling approaches.

1. Introduction

Data mining is a core information technology and has been defined as the major step in the process of Knowledge Discovery in Databases [9]. Many data mining algorithms are query intensive, i.e. numerous queries are initiated on the underlying database system. In the prominent case of multidimensional data spaces, similarity queries, particularly *k-nearest neighbor (k-nn) queries*, are the most important query type [17]. The process of finding the *k*-nearest neighbors of a given query object is a CPU and I/O intensive task and the conventional approach to address this problem is to use some multidimensional index structure [16], [10].

While several sophisticated solutions have been proposed for single *k*-nn queries, the problem of efficiently processing multiple *k*-nn queries issued simultaneously is relatively untouched. However, there are many applications where *k*-nn queries emerge simultaneously. The proximity analysis algorithm proposed in [13], for instance, performs a *k*-nn query for each of the "top-*k*" neighbor objects of any identified cluster in the database. The outlier identification algorithm proposed in [6] performs a *k*-nn query with a high value of *k* for each database object. Another example is the classification of a set of objects which can efficiently be done by using a *k*-nn query for each unclassified object [14]. In all these algorithms *k*-nn queries are processed sequentially, i.e. one after another. However, since many or even all queries do not depend on each other, they can easily be performed simultaneously which offers much more potential for query optimization. In [2], a novel technique for simultaneous query processing called *multiple similarity queries* has been introduced. The authors present a syntactical transformation of a large class of data mining algorithms into a form which uses multiple similarity queries. For the efficient processing of the transformed algorithms, the authors propose to load a page once and immediately process it for all queries which consider this page as a candidate page. It has been shown that this results in a significant reduction of the total I/O cost. By applying the triangle inequality in order to avoid expensive distance calculations, further substantial performance improvements can be achieved. If the underlying access method is scan-based, this scheme is straightforward. However, when performing multiple k-nn queries on top of a multidimensional index structure, the important question emerges in which order the pages should be loaded. This problem has not been addressed in [2]. In this paper, we study the theoretical background of this scheduling problem. In particular, by developing a stochastic model, we find the expected distance to a nearest neighbor candidate located in a considered page to be the key information in order to solve the scheduling problem. Then we propose and evaluate several scheduling techniques which are based on our theoretical results.

The rest of this paper is organized as follows. In section 2, we describe the general processing of multiple k-nn queries. In section 3, we study the theoretical background of the scheduling problem, and in section 4, we propose several scheduling techniques. The experimental evaluation is presented in section 5, and section 6 concludes the paper.

2. Multiple *k*-Nearest Neighbor Queries

We shortly describe the algorithm for processing multiple k-nn queries (cf. Fig. 1) which is based on the HS single k-nn algorithm proposed in [12]. The algorithm starts with m query specifications each consisting of a query point q_i and an integer k_i. For each query an active page list (APL) is created. An APL is a priority queue containing the index pages in ascending order of the minimum distance MINDIST between the corresponding page regions and the query point. In contrast to single query processing, the multiple k-nn algorithm maintains not one APL at a time but m APLs simultaneously. While at least one k-nn query is running, the algorithm iteratively chooses a page P and each query q_i which has not pruned P from its APL_i (i.e. P is still enlisted in APL_i or P was not yet encountered) processes this page immediately. Thus, we only have *one* loading operation even if all queries process P. When processing a page P, the algorithm additionally saves valuable CPU time by avoiding distance calculations which are the most expensive CPU operations in the context of k-nn search. The method $\mathsf{process}(P, q_j)$ does not directly calculate the distances $d(q_j, o_s)$ between objects o_s located on P and a query object q_j, but instead, it first tries to disqualify as many objects o_s as possible by applying the triangle inequality in the following way: if $\left|d(q_i, q_j) - d(q_i, o_s)\right| \geq \mathsf{knn\_dist}(q_j)$, $i < j$, holds, then $d(q_j, o_s) \geq \mathsf{knn\_dist}(q_j)$ is true and the distance $d(q_j, o_s)$ needs not to be calculated. The inter-query distances $d(q_i, q_j)$ are precalculated and $\mathsf{knn\_dist}(q_j)$ denotes the k-nn distance of query q_j determined so far. For all objects which cannot be disqualified, the distance to q_j is computed and stored in the buffer dist_buffer. Note that disqualified distances cannot be used to avoid distance computations of other query objects. Obviously, the method choose_candidate_page() is crucial for the overall performance of the algorithm, since a bad choice results in unnecessary and expensive processing of pages. Thus, it is important to find a schedule which minimizes the total number of processing operations.

```
DB::multiple_knn_queries(q1, ..., qm: query_object,
                                     k1, ..., km: integer)
begin
   precalculate_interquery_distances([q1, ..., qm]);
   for i from 1 to m do create_APL(qi);
   while queries_running([q1, ..., qm]) do
      P = choose_candidate_page([APL1, ..., APLm]);
      for i from 1 to m do APLi.delete(P);
      load(P);
      initialize_dist_buffer();
      for i from 1 to m do
         if isa_candidate_page(P, qi) then
            process(P, qi, ...);
   return ([kNN1, ..., kNNm]);
end.

DB::process(page P, query_object qi, ...)
begin
   foreach object o in P do
      if not avoid_dist_computation(o, qi, dist_buffer) then
         dist_buffer[o][qi] = distance(o, qi);
         if dist_buffer[o][qi] ≤ knn_dist(qi) then
            if isa_directory_page(P) then APLi.insert(o);
            else (* P is a data page *)
               kNNi.insert(o);
               if kNNi.cardinality() > ki then
                  kNNi.remove_last_object();
                  APLi.prune_pages();
   return;
end.
```

Fig. 1. Multiple *k*-nearest neighbor algorithm

3. The Pruning Power Theory

The HS algorithm for processing single *k*-nn queries loads exactly those pages which are intersected by the *k*-nn sphere in ascending order of the MINDIST. In its basic form, the HS algorithm is heavily I/O bound. This behavior changes if we are moving from processing single *k*-nn queries to processing multiple *k*-nn queries. To explain this fact, we have to consider the loading operation load(P) and the processing operation process(P, q_i). When P is a data page, the processing operation determines the distances between the query point q_i and all data points on page P which cannot be avoided by applying the triangle inequality (cf. section 2). In the worst case, the total number of distance computations to be performed for a page P with capacity C_{eff} is $m \cdot C_{\text{eff}}$, i.e. no distance computation could be avoided. The cost for loading P, on the other hand, is independent of the number of queries which actually process this page and therefore remains constant. As a consequence, the algorithm may switch from I/O bound to CPU bound. This switch, however, does not only depend on the number m of simultaneous queries but also on the question, how early the algorithm is able to exclude pages from processing.

3.1 Problem Description

Whenever the distance of the *k*-nn candidate for some query q_i ($1 \le i \le m$) decreases, all pages P_j having a MINDIST larger than the new *k*-nn candidate distance are excluded from being processed for this query (they are pruned off APL_i). If a page is pruned for many queries but a few, the effort of the loading operation must be performed whereas

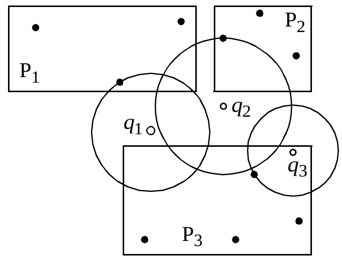

Fig. 2. Example

many processing operations are avoided in advance. Therefore, pruning saves valuable processing time and the algorithm switches back to I/O bound.

The central question we have to address is, what is a suitable order of pages that prunes as many processing operations process(P_j, q_i) as possible? For showing that this is not a trivial task, let us consider the following example in Fig. 2. Applying the HS algorithm for 1-nn queries, query q_1 would first access page P_3 and then P_1. Query q_2 would access the page sequence P_2, P_1, and P_3. Finally, q_3 would only access P_3 and then stop immediately. In multiple query processing, all three pages must be loaded into main memory, because each page is relevant for at least one query. Considering the sequence, however, makes a big difference with respect to the processing operations. For instance, a very bad choice is to start with P_2. After loading P_2, we have to perform all distance calculations between the points on P_2 and all query points since using the triangle inequality cannot avoid any distance calculation. Additionally, the data points found on page P_2 do not exclude any of the remaining pages P_1 and P_3 from any query, because the current 1-nn candidates still have a high distance from the query points. Therefore, the *pruning distances* are bad, or in other words, P_2 does not yield a high *pruning power*. A much better choice is P_3 since once P_3 is loaded and processed, P_1 and P_2 are pruned for query q_3, because the pruning distance of q_3 becomes fairly low. If P_1 is loaded next, P_2 can additionally be pruned for q_1. Finally, P_2 is loaded, but only the distances to q_2 must be determined since P_2 is pruned for all other queries. Thus, we have saved 3 out of 9 processing operations compared to a scheduling sequence starting with page P_2.

3.2 The Pruning Power Definition

This simple example already demonstrates that a good page schedule is essential in order to further improve the processing of multiple *k*-nn queries. Thus, our general objective is to load pages which have a high *pruning power*. We define the pruning power as follows:

Definition 1: Pruning Power

The pruning power of a page P_r is the number of processing operations process(P_s, q_i), $1 \le s \le num\_pages$, $s \ne r$, $1 \le i \le m$, which can be avoided if page P_r is loaded and processed.

According to this definition, the pruning power of a page is exactly known only at the time when the page has been processed. Recall that processing operations can be avoided only if the *k*-nn candidate (and, thus, the pruning distance) for some query changes.

In general, it is not possible to determine in advance whether such a change occurs. Therefore, our goal is first to capture more precisely the conditions under which a page has a high pruning power, and then to develop heuristics which select pages that obey these conditions. The core problem is to determine an expected value r_{prune} for the radius for which the intersection volume contains a number k' of points:

$$r_{\text{prune}} = \int_0^{\infty} \left(r \cdot \frac{\partial}{\partial r} \left(\sum_{x = k'}^{C_{\text{eff}}} \begin{bmatrix} C_{\text{eff}} \\ x \end{bmatrix} \cdot \left(\frac{V_{\text{inters}}(r)}{V_{\text{pagereg}}} \right)^x \cdot \left(1 - \frac{V_{\text{inters}}(r)}{V_{\text{pagereg}}} \right)^{C_{\text{eff}} - x} \right) \right) dr , \qquad (1)$$

where k' is the number of points which are needed to have a set of k points in total. A difficult task here is to determine the intersection volume, which can be solved by a tabulated Montecarlo integration [7]. Although tabulation makes this numerical method possible, the performance is not high enough, since eq. (1) must be evaluated often.

A promising approach to address such problems is to introduce heuristics which inherently follow the optimal solution. When considering single k-nn algorithms proposed in the literature, e.g. [12], [15], we can extract two substantial measures which are associated with the pruning power: the minimum distance MINDIST and the priority of a page in the corresponding APL. Both measures have been successfully used to schedule pages for a single k-nn query. Additionally, it has been shown that other measures, e.g. the minimum of the maximum possible distances from a query point to a face of a considered page region, perform only poorly. Thus, we develop three heuristics in the following section which select pages with a high pruning power on the basis of these two measures, i.e., using the priority of pages in APLs and using the distance MINDIST.

4. Pruning Power Scheduling Strategies

4.1 Average Priority Technique

This heuristic is based on the observation that a page can only have a high pruning power if it has a high local priority in several APLs. If the page has a low priority in most of the APLs, it is obviously very unlikely that this page yields a high pruning power. When selecting the next page to process, for each page P_j the ranking numbers $\text{RN}(\text{APL}_i, P_j)$ in all queues APL_i are determined and summed up. If, for instance, P_1 is at the first position for q_1, at the fifth position for q_2 and at the 13th position for q_3, we get a cumulated ranking number of 19 for P_1. This number is divided by the number (3 in this example) of APLs that contain P_1 in order to form an average and to avoid the underestimation of pages that are already pruned for many queries. The page with the lowest average ranking $\bar{P}$ is selected to be loaded and processed next. With npq_i we denote the number of priority queues which contain page P_i.

$$\bar{P} = \text{MIN}\left(\frac{1}{npq_1} \cdot \sum_{i=1}^{m} \text{RN}(\text{APL}_i, P_1), \ldots, \frac{1}{npq_s} \cdot \sum_{i=1}^{m} \text{RN}(\text{APL}_i, P_s) \right) \qquad (2)$$

4.2 Average Distance Technique

The average distance strategy is similar to the average priority heuristic. The focus of this approach is not the position of a page in the APLs, but directly the minimum distance MD(q_i, P_j) between a page region P_j and a query point q_i. The motivation is the observation that the pruning power of a page is monotonically decreasing with increasing distance to the query point, i.e. when a page is located far away from most query points, this page probably has a low pruning power. For each page, the distances to all query points (except those for which the considered page is already pruned) are determined and summed up. Again, the average is formed by dividing the cumulated distance by the number of all queries that did not yet prune the page. The page with the least average distance $\overline{P}$ is selected for being processed next.

$$\overline{P} = \mathrm{MIN}\left(\frac{1}{npq_1}\cdot\sum_{i=1}^{m}\mathrm{MD}(q_i, P_1), \ldots, \frac{1}{npq_s}\cdot\sum_{i=1}^{m}\mathrm{MD}(q_i, P_s)\right) \tag{3}$$

4.3 Maximum Priority Technique

The motivation for this strategy is the observation that pages with maximum priority (i.e. they are at the top position) in one or more priority queues often have a high pruning power even if they have very low priority for other queries. The reason is that pages with maximum priority generally are more likely to contain one of the *k* nearest neighbors than the pages on the following positions. Therefore, a page which is at the first position for few queries may outperform a page that is at the second position for many queries. Like the average priority technique, the maximum priority technique determines the ranking of all pages with respect to the position in the priority queues. In contrast to the average priority technique, it counts the number MPC(APL_1, ..., APL_m, P_j) of queries q_i, for which the page P_j has the maximum priority. The page yielding the highest count $\overline{P}$ is selected as the next page to be processed. When two or more pages have an equal count, we consider position-two (and subsequent) counts as a secondary criterion.

$$\overline{P} = \mathrm{MAX}\ (\mathrm{MPC}(\mathrm{APL}_1, \ldots, P_1), \ldots, \mathrm{MPC}(\mathrm{APL}_1, \ldots, P_s)) \tag{4}$$

5. Experimental Evaluation

In order to determine the most efficient scheduling technique we performed an extensive experimental evaluation using the following databases:

- *Synthetic database*: 1,600,000 8-*d* random points following a uniform distribution.
- *CAD database*: 16-*d* Fourier points corresponding to contours of 1,280,000 industrial parts used in the S3-system [4].
- *Astronomy database*: 20-*d* feature vectors of 1,000,000 stars and galaxies which are part of the so-called Tycho catalogue [11].

All experiments presented in this section were performed on an Intel Pentium II-300 workstation under Linux 6.0. The index structure we used was a variant of the X-tree where the directory consists of one large supernode. We used a block size of 32 KBytes for the X-tree and the cache size was set to 10% of the X-tree size.

For index structures with a hierarchically organized directory, the proposed scheduling techniques can be applied as dynamic approaches or as hybrid approaches (sequences of static schedules). However, with the directory consisting of one large supernode, we are able to apply the techniques in a purely static way: Since we can completely construct the APL once a query point is provided, we can also determine a page schedule

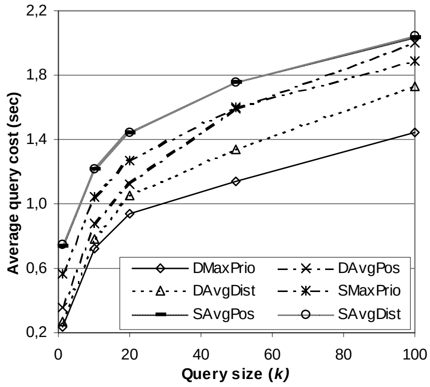

Fig. 3. Query size on the CAD database

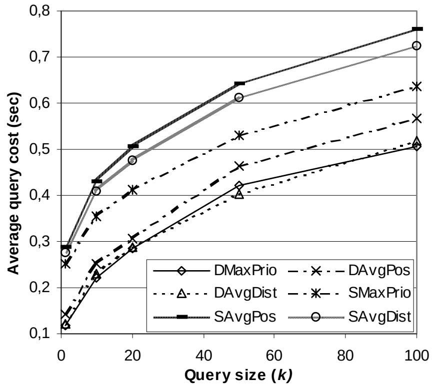

Fig. 5. Query size on the synthetic database

before we start query processing. While this is straightforward for the average priority and the average distance technique, we have to slightly modify the maximum priority technique. First, we determine all pages having a maximum priority and sort them with respect to their counts. Assuming that those pages are already processed, we consider all pages located on the second position of any APL as pages with maximum priority and again sort them. This step iterates until all pages are enlisted in the schedule. Additionally, for all static approaches we check if a chosen page is still needed by any query before loading the page. We experimentally evaluate the following page scheduling techniques (cf. section 4):

- Static and Dynamic Average Priority (denoted as *SAvgPos* and *DAvgPos*)
- Static and Dynamic Average Distance (denoted as *SAvgDist* and *DAvgDist*)
- Static and Dynamic Maximum Priority (denoted as *SMaxPrio* and *DMaxPrio*)

We first investigate the effect of the query size using the CAD database. The maximum number *m* of multiple *k*-nn queries in the system is set to 20 and we make the simplifying assumption that at any time there are enough queries in a queue waiting to be processed to load the system completely. The query size *k* varies and Fig. 3 depicts the average query cost including the CPU and I/O cost. We can observe that *DMaxPrio* clearly outperforms all other scheduling techniques for all values of *k*. Compared to the second best technique *DAvgDist*, the *DMaxPrio* approach exhibits 85% of the average query cost for $k = 1$ and 83% of the average query cost for $k = 100$. Considering the *SAvgPos* and the *SAvgDist* approaches (their performance plots are almost identical), the average query cost of *DMaxPrio* is only 32% (70%) of the corresponding average query cost for 1- (100-) nearest neighbor queries. All dynamic techniques outperform the static approaches up to $k = 50$. For $k > 50$, *SMaxPrio* starts to outperform the *DAvgPos* approach. With increasing *k*, the performance gain of *DMaxPrio* compared to the other techniques decreases. The reason is that with increasing query size, the distance to the *k*-nn (pruning distance) increases and fewer distance calculations can be avoided by applying the triangle inequality. However, even for high values of *k* the *DMaxPrio* approach saves 12% - 30% of the query cost compared to the other techniques.

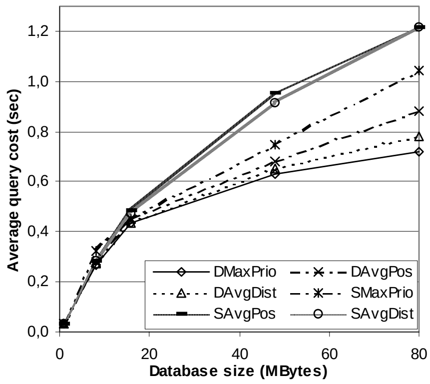

Fig. 6. Database size on the CAD database

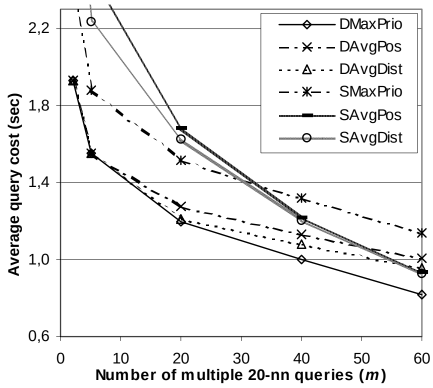

Fig. 7. # of mult. queries (Astronomy db)

The same experiment is performed on the synthetic database (cf. Fig. 5). As before, the *DMaxPrio* approach yields the best overall performance and leads to an average query cost of 36% - 59% for $k = 1$ and 7% - 34% for $k = 100$ of the average query cost of the other scheduling techniques except for *DAvgDist*. In this experiment, the *DAvgDist* approach shows a comparative performance as *DMaxPrio* and even outperforms *DMaxPrio* for $k = 50$. For all values of k, the dynamic techniques provide a much better performance than the static techniques. Considering only the static approaches, we observe that *SMaxPrio* outperforms *SAvgPos* and *SAvgDist*.

Next, we analyze the impact of the database size on the scheduling techniques. We used the CAD database and increased the number of Fourier points from 12,800 up to 1,280,000. We kept the maximum number m of multiple k-nn queries at 20 and performed 10-nn queries (cf. Fig. 6). For small database sizes, all scheduling techniques show similar performance. When increasing the database size, this situation changes drastically: For database sizes larger than 16 MBytes, the performance of *SAvgPos* and *SAvgDist* degenerates whereas *DMaxPrio* and *DAvgDist* show a comparatively moderate increase of the average query cost (*DMaxPrio* again outperforms *DAvgDist*). This can be explained by the following observation: With increasing database size, the average length of the APLs also increases and the static approaches more and more suffer from the lack of information resulting from processing candidate pages. The average query cost of *SMaxPrio* shows an acceptable increase up to 48 MBytes. For 80 MBytes, however, also this approach exhibits poor performance. Considering the dynamic scheduling techniques, *DAvgPos* has the worst performance for large database sizes.

Since dynamic approaches generally introduce some computational overhead for determining the page schedule, we also investigated the system parameter m which typically depends on hardware aspects and on the underlying application. We used the astronomy database and performed 20-nn queries while increasing the maximum number m of multiple queries (cf. Fig. 7). For all scheduling techniques, the average query cost clearly decreases with increasing m which underlines the effectiveness of the multiple query approach. Again, *DMaxPrio* yields the best overall performance (it outperforms all other techniques for $m \geq 5$). An important result is the following observation: While

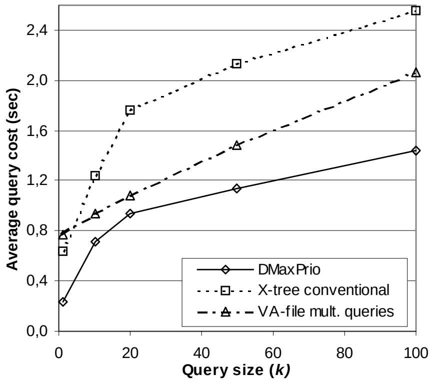

Fig. 8. Effectiveness of the multiple query approach on the CAD database

for small values of m the dynamic approaches obviously outperform the static approaches, we observe that *SAvgPos* and *SAvgDist* outperform *DAvgPos* and *DAvgDist* for $m = 60$. The reason for this result is the increasing cost of dynamically calculating the scheduling criterion since the number of APLs that must be analyzed is directly proportional to the number of queries in the system. The *DMaxPrio* approach, on the other hand, exhibits excellent performance even for high values of m due to the fact that the decision criterion can be evaluated at almost no extra cost, since in average only the top elements of each APL have to be analyzed.

The objective of our last experiment is to show the efficiency of our approach in general. We compared the most efficient pruning power scheduling technique (*DMaxPrio*) with the multiple queries technique using an efficient variant of the linear scan, namely the VA-file [17], and we compared it with conventional query processing using the X-tree. For this experiment, we used the CAD database, set m to 20 and varied the query parameter k from 1 to 100. The average query cost with respect to k is depicted in Fig. 8. Our new scheduling technique clearly outperforms the conventional X-tree query processing by speed-up factors ranging from 2.72 for 1-nn queries to 1.78 for 100-nn queries. Considering the VA-file using the multiple query scheme (*VA-file mult. queries*), we can observe that the average query cost of *DMaxPrio* is less than the average query cost of the multiple query VA-file for all values of k. However, this does not hold for all scheduling techniques. For instance, using *DAvgPos* or a static approach (e.g. *SMaxPrio*) for the page scheduling, the multiple query VA-file outperforms the multiple query X-tree already for $k > 10$ (compare with Fig. 3). This result underlines the importance of finding an efficient and robust scheduling technique in order to maximize the performance improvement resulting from the multiple query scheme.

6. Conclusions

In this paper, we have studied the problem of page scheduling for multiple k-nn query processing prevalent in data mining applications such as proximity analysis, outlier identification or nearest neighbor classification. We have derived the theoretic foundation and found that the pruning power of a page is the key information in order to solve the scheduling problem. The pruning power results from the distance between the *k*-nn candidates located in a data page and the query points. We have proposed several scheduling algorithms which base on the pruning power theory. An extensive experimental evaluation demonstrates the practical impact of our technique. For future work, we plan to analyze our technique in parallel and distributed environments.

References

1. Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *'A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces'*, Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, Tucson, USA, 1997, pp. 78-86.
2. Braunmüller B., Ester M., Kriegel H.-P., Sander J.: '*Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases*', Proc. 16th Int. Conf. on Data Engineering, San Diego, USA, 2000, pp. 256-267.
3. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., Sander J.: '*Independent Quantization: An Index Compression Technique for High-Dimensional Spaces*', Proc. Int. Conf. on Data Engineering, San Diego, USA, 2000, pp. 577-588.
4. Berchtold S., Kriegel H.-P.: *'S3: Similarity Search in CAD Database Systems'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, USA, 1997, pp. 564-567.
5. Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data'*, Proc. Conf. on Very Large Data Bases, Mumbai, India, 1996, pp. 28-39.
6. Breunig M. M., Kriegel H.-P., Ng R., Sander J.: 'OPTICS-OF: Identifying Local Outliers', Proc. Conf. on Principles of Data Mining and Knowledge Discovery, Prague, 1999, in: Lecture Notes in Computer Science, Springer, Vol. 1704, 1999, pp. 262-270.
7. Böhm C.: *'Efficiently Indexing High-Dimensional Data Spaces'*, Ph.D. thesis, University of Munich, Munich, Germany, 1998.
8. Friedman J. H., Bentley J. L., Finkel R. A.: '*An Algorithm for Finding Best Matches in Logarithmic Expected Time*', ACM Transactions on Mathematical Software, Vol. 3, No. 3, 1977, pp. 209-226.
9. Fayyad U. M., Piatetsky-Shapiro G., Smyth P.: '*From Data Mining to Knowledge Discovery: An Overview*', Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 1 - 34.
10. Gaede V., Günther O.:'*Multidimensional Access Methods*', ACM Computing Surveys, Vol. 30, No. 2, 1998, pp.170-231.
11. Høg E. et al.: "*The Tycho Catalogue*", Journal of Astronomy and Astrophysics, Vol. 323, 1997, pp. L57-L60.
12. Hjaltason G. R., Samet H.: '*Ranking in Spatial Databases*', Proc. Int. Symp. on Large Spatial Databases, Portland, USA, 1995, pp. 83-95.
13. Knorr E.M., Ng R.T.: '*Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining*,' IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 884-897.
14. Mitchell T. M.: '*Machine Learning*', McGraw-Hill, 1997.
15. Roussopoulos N., Kelley S., Vincent F.: '*Nearest Neighbor Queries*', Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, USA, 1995, pp. 71-79.
16. Samet H.: '*The Design and Analysis of Spatial Data Structures*', Addison-Wesley, 1989.
17. Weber R., Schek H.-J., Blott S.: '*A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces*', Proc. Int. Conf. on Very Large Data Bases, New York, USA, 1998, pp. 194-205.