A Cost Model for Reverse Nearest Neighbor Query Processing on R-trees Using Self Pruning

Felix Borutta[1], Peer Kröger[2], and Matthias Renz[2]

[1] Institute for Computer Science
Ludwig-Maximilians-Universität München, Germany
borutta@dbs.ifi.lmu.de
[2] Institute for Computer Science
Christian-Albrechts-Universität zu Kiel, Germany
{pkr, mr}@isdm.informatik.uni-kiel.de

Abstract. In this short paper, we propose the first cost model for a class of index structures designed for reverse nearest neighbor (RNN) search, so-called self pruning approaches. These approaches use estimations of the nearest neighbor distances of database objects for pruning. We will particularly detail our cost model for R-Trees but our concepts can easily applied to any tree-like index structure that implements a self pruning strategy. Our cost model estimates the number of disk accesses of a given RNN query and, thus, allows to predict the required I/O costs in any hardware environment. We further explore three variants regarding the trade-off between estimation accuracy and model efficiency/storage overhead. Preliminary experiments on synthetic data confirm that the estimations are accurate compared to the exact query costs.

1 Introduction

Reverse nearest neighbor (RNN) queries are prevalent in many practical applications since they determine the set of data objects influenced by the query. Specifically, an RNN query retrieves those objects from the database having the query as one of their nearest neighbors (NNs). Variants of this basic query introduce the parameter k specifying the number of NNs that are considered (i.e., the query must be among the kNNs of a true hit), and/or a distinction between query set and answering set (so-called bichromatic scenario compared to the "normal" case that is referred to as mono-chromatic).

Beside a plethora of index structures and algorithms especially designed to optimize RNN query processing, to the best of our knowledge, no work has been done for estimating the costs of these approaches so far. Predicting the costs for processing a given query is mandatory for (relational) query optimizers. A cost model allows to generate efficient query plans and enables effective scheduling. Thus, this work is a first step towards the practical use of the existing query processing algorithms and their respective data structures in real database systems.

In this short paper, we sketch the idea for a general cost model for RNN queries for a rather general class of query processing algorithms. Existing algorithms for RNN query processing can be classified according to the applied strategy of pruning objects from the search space. *Self pruning* approaches typically rely on special index structures that

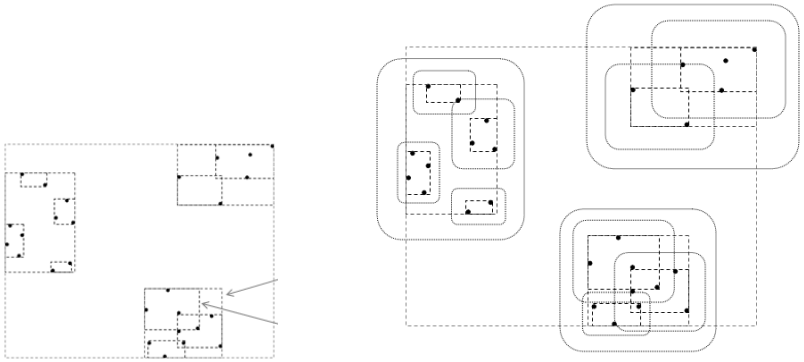
usually offer a higher selectivity and, hence, are more efficient. In turn, these methods are usually less flexible than *mutual pruning* approaches in terms of query parametrization and database updates. Since both pruning strategies are rather different paradigms, we focus on self pruning methods only. We explore a general way to estimate the number of index pages that need to be accessed for a given query which allows to predict the I/O costs on any hardware environment. The cost model will be explained using a concrete instance of self pruning methods, the RdNN tree [1] which basically uses a member of the R-Tree family. However, we want to emphasize that our general idea is independent of the used index and can be adapted to any tree-like index implementing a self pruning method very easily. The reason for this is that the only basic assumption of our cost model is that information on the size of the page regions, e.g. minimum bounding rectangles (MBRs) in case of an R-Tree, of index nodes is know. This information typically contains only a very few numbers and can easily be materialized in the cache.

The remainder is organized as follows. Section 2 gives an overview of related work. In Section 3, we explain our new cost model for RNN query processing using self pruning methods. Some preliminary experiments are presented in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

Self pruning approaches like [2, 1] are usually designed on top of a tree-like index structure. They are based on the observation that if the distance between any database object o and the query q is smaller than the kNN distance of o, o is part of the result set, i.e., a RNN of q. Otherwise, o can be pruned. Consequently, self pruning approaches try to exactly compute or (conservatively) approximate the kNN distance of each index entry e. If this estimate is smaller than the distance of e to the query q, then e can be pruned. Often, self pruning approaches simply pre-compute kNN distances of database points and propagate maxima of these distances to higher level index nodes for pruning. The major limitation of these approaches is that the pre-computation is time and memory consuming and less flexible to database updates. These methods are usually limited to one specific or very few values of k. Approaches like [3–9] try to overcome these limitations by using approximations of kNN distances (for any k) but this yields an additional refinement overhead – or only approximate results. *Mutual pruning approaches* such as [10, 11] use other points to prune a given index entry e. For instance, [11] iteratively constructs Voronoi hyper-planes around the query q from a nearest neighbor ranking w.r.t. q. Points and index entries that are beyond k Voronoi hyper-planes w.r.t. q can be pruned. Mutual pruning approaches need an additional refinement of candidates (i.e., a kNN query for not pruned objects) to compute the final results.

As mentioned above, to the best of our knowledge, there are no cost models for RNN algorithms proposed so far. However, there are a lot of cost models for other query types on R-Trees, including NN queries (e.g. [12–14]) and spatial join queries (e.g. [15, 16]). Closest to our work is the method for range queries independently proposed by [17] and [18]. Both approaches assume that the MBR of each node in the underlying R-Tree is given and estimate the disc accesses using the concept of the Minkowski sum. We will revisit details on this model later.



(a) (a) R*-Tree. (b) (b) RdNN-Tree.

Fig. 1: Visualization of an RdNN-Tree (b) as an extension of the R*-Tree (a).

3 A Cost Model for Self Pruning Approaches

For a positive integer k and a query object q, a k-NN query retrieves the set $NN(q,k)$ including those k points having the smallest distance $dist(.,.)$ to q. In case of ties, this set may have more than k elements. The distance between p and its kNN is called kNN-distance (denoted by $kNNdist(p)$) of p. A RkNN query with query object q can be defined as those objects having q as one of their kNN, i.e., $RNN(q,k) = \{o \in DB | o \in NN(q,k)\}$. If k is clear from context, we omit it and use NN, $NNdist(.)$, and RNN instead of kNN, $kNNdist(.)$, and RkNN.

The basic observation behind self-pruning approaches is that an object p qualifies for a given RNN query if and only if $dist(p,q) \leq NNdist(p)$. Thus, materializing (exact or approximate) NN-distances of all database objects provides a powerful and very selective pruning possibility. Self-pruning approaches use any conventional index, e.g. an R*-tree (as in the RdNN-Tree [1]), to organize the data objects but additionally stores the pre-computed NN-distances. For a data page of the index containing a set of database objects, an approximation of the NN-distances of all points of this page needs to be derived and this approximation needs to be conservative for producing exact results. This can easily be done by aggregating the maximum of all NN-distances in that page. For directory pages of a tree-based index containing child pages each associated with a NN-distance estimate of its corresponding sub-tree, the procedure is similar: the maximum NN-distance of all child nodes need to be aggregated. Thus, each node N of the index aggregates the maximum NN-distance of all objects represented by N. The extension of an R*-Tree to an RdNN-Tree is visualized in Figure 1. The aggregated maximum NN-distance of each node N, denoted by $NNdist(N)$, is visualized as box with rounded corners around the corresponding page regions (PRs). These distances can be used during query processing: node N may contain a true hit if for the minimum distance $MINDIST$ between an object q and the PR of N it holds: $MINDIST(q,N) \leq NNdist(N)$. In this case, the subtree of N needs to be traversed. Else, node N can be pruned.

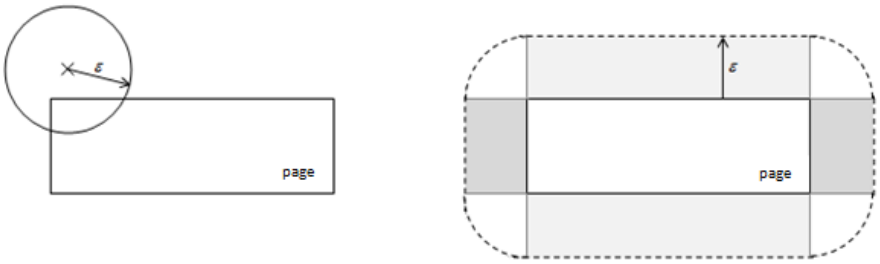


Fig. 2: Range query with radius ε and the page region of an arbitrary index page (left) and the corresponding spatially extended region a.k.a. Minkowski sum (right).

We will show our ideas using the RdNN, assuming that the PRs of the underlying index are minimum bounding rectangles (MBRs), the distance approximations of directory nodes are conservative, and the index is tree-based in the following but our ideas can be extended to any other shapes, approximations, and indexes. The basic observation of our approach is that the situation depicted in Figure 1(b) is related to the cost model for (aka ε-)range queries [17, 18] which estimates the probability of an intersection between the query (circle around q with radius ε) and the PRs of each level of the tree, i.e., the probability that the corresponding subtree needs to be traversed. For this purpose, the PRs are spatially extended by the radius ε as it is done in self-pruning approaches. However, for ε-range queries the spatial extend of all PRs is fixed to ε, while for the RdNN-Tree each PR has its own aggregated maximum NN-distance specifying the spatial extend.

Our cost model basically estimates for each node in the index tree the probability of being traversed when a given query is launched in order to determine the average amount of nodes that need to be accessed. It is based on the same assumptions claimed in [17, 18]; the most basic assumption is that we have information on the PRs of each node in the index. We first start with revisiting the cost model of range queries.

For a range query with radius ε, the access probability of a node N with page region $N.Reg$ [3] is given by

$$PR(Access(N)) = \frac{Vol(N.Reg \textit{ spatially extended by } \varepsilon)}{Vol(\textit{data space})},$$

where $Vol(.)$ computes the volume of a region. The page region $N.Reg$ spatially extended by the query radius ε correspond to a bounding box with rounded corners such that the edges have distance ε to the original page region $N.Reg$. This region is known as the Minkowski sum. The idea is visualized in Figure 2.

In order to be able to compute the probability of accessing a given node N, we need to compute the volume of the Minkowski sum of the page region $N.Reg$ of N and ε (corresponding to the numerator in the above formular). Let $N.e$ be the edge length of the page region of N, then the volume of the Minkowski sum of $N.Reg$ and ε is

[3] As mentioned above, our method is not restricted to the exact geometry of page regions.

$$Vol(N.Reg \textit{ spatially extended by } \varepsilon) = Vol_{Minkowski}(N.Reg, \varepsilon) =$$

$$\sum_{0 \le i \le d} \binom{d}{i} \cdot 2^{d-i} \cdot N.e^i \cdot \frac{V(\textit{Sphere}_{(d-i)}(\varepsilon))}{2^{d-i}} = \sum_{0 \le i \le d} \binom{d}{i} \cdot N.e^i \cdot V(\textit{Sphere}_{(d-i)}, \varepsilon).$$

where $\textit{Sphere}_{(d-i)}(\varepsilon)$ is a $(d-i)$-dimensional sphere of radius ε. The volume of this sphere, $Vol(\textit{Sphere}_{(d-i)}(\varepsilon))$, can be computed using the Gamma function:

$$Vol(\textit{Sphere}_{(d-i)}(\varepsilon)) = \frac{\pi^{\frac{d-i}{2}} \cdot \varepsilon^{d-i}}{\Gamma(\frac{d-i}{2}+1)}.$$

The Minkowski volume can be used to calculate the volume of any node N of the index and can be used to compute the probability that N needs to be accessed. In order to estimate the costs for the entire index, we need to determine the access probabilities for all index nodes. For that purpose, we need the edge lengths $N.e$ of all index nodes N. One way to get this is to materialize these values which is typically not a significant overhead and can often even be hold in main memory. If the overhead of storing and updating this information is too large, [17] and [18] offer a way to estimate these values. This estimation is done level-wise: the number of nodes N_i on level i of the tree, denoted by $Card(N_i)$, can recursively be obtained from the average storage utilization. Under the assumption that the MBR of each node N_i is a hyper-cube with equal edge length $N_i.e$ and that its expected volume is $Vol(N_i) = Vol(\textit{data space}) \,/\, Card(N_i)$ we can estimate the average edge length of N_i as

$$N_i.e = \sqrt[d]{\frac{Vol(\textit{data space})}{Card(N_i)}}.$$

Thus, the total number of index nodes (i.e., pages) accessed while processing an ε-range query can be approximated as:

$$\textit{\# page accesses} = \sum_{i=1}^{\text{indexheight}} Card(N_i) \cdot Vol_{Minkowski}(\sqrt[d]{\frac{Vol(\textit{data space})}{Card(N_i)}}, \varepsilon).$$

For the transformation of this model from range queries to RNN queries we first explore the relationship between these two query types. Intuitively, range queries retrieve those objects o that are enclosed in a sphere centered at the query object q having the query range ε as radius, i.e., $dist(q,o) \le \varepsilon$. When a self pruning approach is implemented using pre-computed NN-distances, RNN queries retrieve those objects o that are the center of a sphere which has the NN-distance of o as radius and in which q is enclosed, i.e., $dist(q,o) \le NNdist(o)$. It should be mentioned, however, that this relationship cannot be used to process RNN queries like range queries in general. Only the aggregation and materialization of the NN-distances in the index as proposed in the literature enables to build this relationship.

During the processing of a range query, a page must be accessed if its page region (e.g. MBR) intersects with the query range, i.e., the sphere with radius ε centered at q.

For a RNN query, a page must be accessed if the Minkowski sum of its page region and its maximum NN-distance includes the query. Thus, for any node N in the index, we need to use its maximum NN-distance, $NNdist(N)$ to compute the Minkowski volume:

$$Vol_{Minkowski}(N.e, NNdist(N)) = \sum_{0 \leq i \leq d} \binom{d}{i} \cdot N.e^i \cdot Vol(\textit{Sphere}_{(d-i)}(NNdist(N))).$$

The edge length can be approximated as described above. The remaining challenge now is that while for range queries, the radius ε is fixed in all Minkowski volumes, the aggregated maximum NN-distances of the nodes on level index i can be rather different. In the following, we propose three variants to solve this.

Variant 1: The first variant accounts for the variation of NN-distances and sums up all Minkowski volumes of all index nodes. Note that this requires to have access to all NN-distance values of all nodes N_i on all levels i of the index which could be materialized (for small data sets even in the cache). The number of page accesses is

$$\textit{\# page accesses} = \sum_{i=1}^{\text{indexheight}} \sum_{n \in N_i} Vol_{Minkowski}(\sqrt[d]{\frac{Vol(\textit{data space})}{Card(n)}}, NNdist(n)).$$

Variant 2: If the index is large and pre-computing/materialization of NN-distances for all index entries is not an option, the necessary information needs to be fetched from disc involving a huge overhead of I/O accesses (all nodes of the tree need to be accessed). We can circumvent this by taking the NN-distance of the root $Root$ of the index which is the maximum NN-distances of all data objects. Obviously, this comes to the cost of decreasing the accuracy of the estimation. If M is the number of all nodes of the index, then, we can estimate the number of page accesses by

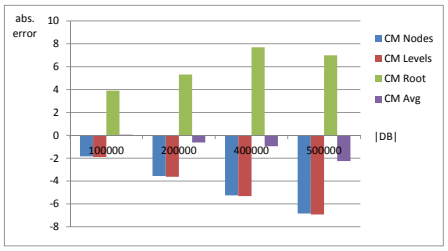
$$\textit{\# page accesses} = M \cdot Vol_{Minkowski}(\sqrt[d]{\frac{Vol(\textit{data space})}{Card(N_i)}}, NNdist(Root)).$$

Variant 3: The variants discussed above basically trade-off the accuracy of the estimation and the costs for obtaining the estimation (in terms of storage overhead or, if the required information needs to be fetched from disc, in terms of time). As a compromise we propose to aggregate the average NN-distances for each index level which causes much less overhead to maintain and materialize than in Variant 1 but should give better estimates than Variant 2. The average NN-distance of all nodes N_i of level i is

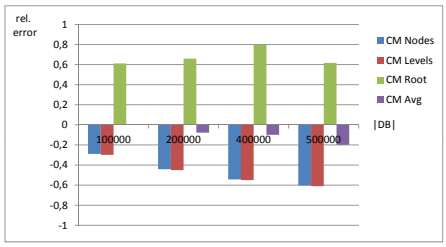
$$NNdist_i^{avg} = \frac{\sum_{N_i} NNdist(N_i)}{Card(N_i)}.$$

Then the number of page accesses can be calculated as

$$\textit{\# page accesses} = \sum_{i=1}^{\text{indexheight}} Card(N_i) \cdot Vol_{Minkowski}(\sqrt[d]{\frac{Vol(\textit{data space})}{Card(N_i)}}, NNdist_i^{avg}).$$

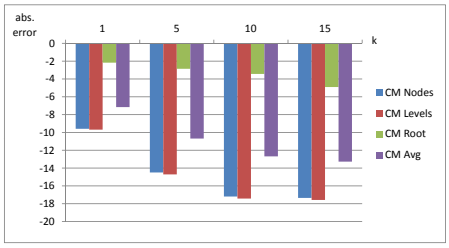


(a) Clustered data: absolute error.

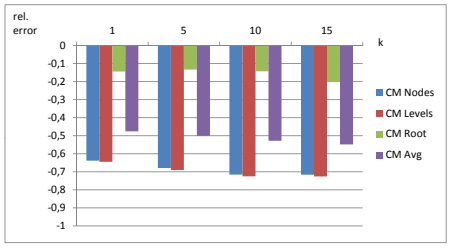


(b) Clustered data: relative error.

Fig. 3: Accuracy of the estimation w.r.t. varying data size.



(a) Clustered data: absolute error.



(b) Clustered data: relative error.

Fig. 4: Accuracy of the estimation w.r.t. varying k.

4 Preliminary Empirical Study

We evaluate how accurate our model can estimate the real page accesses for a given query and report absolute and relative estimation errors for all three variants discussed above for the RdNN-tree implementation of ELKI [19] with a page size of 8K. We used a 3D synthetic data sets with 10 clusters of equal size each following a Gaussian distribution with random mean and standard deviation and an additional 10% uniformly distributed noise. In all runs, we used 50% of the database points and another 50% of randomly generated points as query objects and averaged the results.

Figure 3 displays the accuracy w.r.t. the database size ($k = 1$). Both the absolute and relative error is considerably small and stable and only grows slowly with increasing database size. Variant 2 that only considers the root node overestimates the costs while all other estimations are conservative. Figure 4 depicts the accuracy of the model variants w.r.t. the query parameter k. The database size is fixed at 200,000 points. Here, all estimates are conservative. With increasing k, the error increases most likely because the kNN distance exponentially contributes to the Minkowski volume.

We also conducted first experiments on the impact of the data dimensionality (omitted due to space limitations). The results show low effects of the data dimensionality as long as it is moderate (> 20), but we assume that a potential break-down of the index may not be accommodated adequately in the cost model.

5 Discussion

In this short paper, we present a first cost model for RNN query processing algorithms using self pruning. We described three different variants that explore the trade-off between estimation accuracy and efficiency/storage overhead. The cost model estimates the number of page accesses for RNN queries on a given index and, thus, is independent of any hardware environment. Our preliminary results confirm that the accuracy of the cost model is promising in a broad range of settings.

References

1. Yang, C., Lin, K.I.: An index structure for efficient reverse nearest neighbor queries. In: Proc. ICDE. (2001)
2. Korn, F., Muthukrishnan, S.: Influenced sets based on reverse nearest neighbor queries. In: Proc. SIGMOD. (2000)
3. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In: Proc. SIGMOD. (2006)
4. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE TKDE **18** (2006) 1239–1252
5. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Approximate reverse k-nearest neighbor search in general metric spaces. In: Proc. CIKM. (2006)
6. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor estimation. In: Proc. BTW. (2007)
7. Figueroa, K., Paredes, R.: Approximate direct and reverse nearest neighbor queries, and the k-nearest neighbor graph. In: Proc. SISAP. (2009)
8. Casanova, G., Englmeier, E., Houle, M.E., Kröger, P., Nett, M., Schubert, E., Zimek, A.: Dimensional testing for reverse k-nearest neighbor search. In: Proc. VLDB. (2017)
9. Berrendorf, M., Borutta, F., Kröger, P.: k-distance approximation for memory-efficient rknn retrieval. In: Proc. SISAP. (2019)
10. Singh, A., Ferhatosmanoglu, H., Tosun, A.S.: High dimensional reverse nearest neighbor queries. In: Proc. CIKM. (2003)
11. Tao, Y., Papadias, D., Lian, X.: Reverse kNN search in arbitrary dimensionality. In: Proc. VLDB. (2004)
12. Papadopoulos, A., Manolopoulos, Y.: Performance of nearest neighbor queries in r-trees. In: Proc. ICDT. (1997)
13. Berchtold, S., Böhm, C., Keim, D.A., Kriegel, H.P.: A cost model for nearest neighbor search in high-dimensional data space. In: Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1997, Tucson, Arizona. (1997)
14. Korn, F., Pagel, B., Faloutsos, C.: On the dimensionality curse and the self-similarity blessing. IEEE TKDE (2001)
15. Huang, Y., Jing, N., Rundensteiner, E.: A cost model for estimating the performance of spatial joins using r-trees. In: Proc. SSDBM. (1997)
16. Böhm, C., Kriegel, H.P.: A cost model and index architecture for the similarity join. In: Proc. ICDE. (2001)
17. Kamel, I., Faloutsos, C.: On packing r-trees. In: Proc. CIKM. (1993)
18. Pagel, B.U., Six, H., Toben, H., Widmayer, P.: Towards an analysis of range query performance. In: PODS. (1993)
19. Achtert, E., Kriegel, H.P., Zimek, A.: ELKI: a software system for evaluation of subspace clustering algorithms. In: Proc. SSDBM. (2008)