Appeared in proceedings of TAMC 2008

Search Space Reductions for Nearest-Neighbor Queries

Micah Adler[1] and Brent Heeringa[2]

[1] Department of Computer Science, University of Massachusetts, Amherst
140 Governors Drive Amherst, MA 01003
[2] Department of Computer Science, Williams College, Williamstown, MA, 01267
`micah@cs.umass.edu, heeringa@cs.williams.edu`

Abstract. The vast number of applications featuring multimedia and geometric data has made the R-tree a ubiquitous data structure in databases. A popular and fundamental operation on R-trees is nearest neighbor search. While nearest neighbor on R-trees has received considerable experimental attention, it has received somewhat less theoretical consideration. We study pruning heuristics for nearest neighbor queries on R-trees. Our primary result is the construction of non-trivial families of R-trees where k-nearest neighbor queries based on pessimistic (i.e. min-max) distance estimates provide exponential speedup over queries based solely on optimistic (i.e. min) distance estimates. The exponential speedup holds even when $k = 1$. This result provides strong theoretical evidence that min-max distance heuristics are an essential component to depth-first nearest-neighbor queries. In light of this, we also consider the time-space tradeoffs of depth-first versus best-first nearest neighbor queries and construct a family of R-trees where best-first search performs exponentially better than depth-first search even when depth-first employs min-max distance heuristics.

1 Introduction

Nearest neighbor queries on the R-tree play an integral role in many modern database applications. This is due in large part to the prevalence and popularity of multimedia data indexed geometrically by a vector of features. It is also because nearest neighbor search is a common primitive operation in more complex queries [1].

Although the performance of nearest neighbor search on R-trees has received some theoretical consideration (e.g., [2, 3]), its increasing prominence in today's computing world warrants even further investigation. The authors of [1] note that three issues affect the performance of nearest neighbors on R-trees:

- the order in which children are visited,
- the traversal type, and
- the pruning heuristics.

We show that at least two of these — traversal type and pruning heuristics — have a quantitatively profound impact on efficiency. In particular we prove the following:

1. There exists a family of R-trees where depth-first k-nearest neighbor search with pessimistic (i.e. min-max) distance pruning performs exponentially better than optimistic (i.e. min) distance pruning alone. This result holds even when $k = 1$.
2. There exists a family of R-trees where best-first k-nearest neighbor queries perform exponentially better than depth-first nearest neighbor queries even when the depth-first search uses both optimistic and pessimistic pruning heuristics. This result also holds when $k = 1$.

Our first result provides strong theoretical evidence that pruning strategies based on pessimistic distance estimates are valuable in depth-first nearest neighbor queries. These results rely on subtle changes to existing algorithms. In fact, without these nuanced changes, the exponential speedup may completely disappear.

Our second result deals with the known time efficiency benefits of best-first nearest neighbor algorithms over depth-first nearest neighbor algorithms. Given our first result, it is natural to ask whether pessimistic distance pruning closes part of the time efficiency gap. We answer this question in the negative through several general constructions. Still, the benefit of pessimistic pruning strategies should not be overlooked. They provably enhance the time-efficiency of the already space-efficient depth-first nearest neighbor queries.

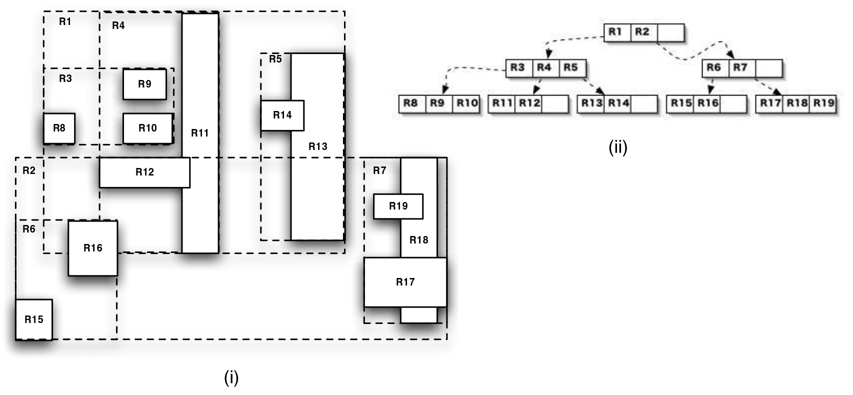

Fig. 1. (i) A collection of spatial objects (solid lines) and their hierarchy of minimum bounding rectangles (dashed lines). (ii) An R-tree for the objects in (i).

Such algorithms still play an increasingly prominent role in computing given the frequency and demand for operations on massive data sets.

The outline of this paper is as follows: In sections 2 and 3 we briefly review definitions for R-trees, MinDist, MinMaxDist, and the three common pruning heuristics employed in depth-first nearest neighbor queries. Sections 4 and 5 describe our R-tree constructions and prove the power of pessimistic pruning. Section 6 discusses the time-space tradeoffs of best-first versus depth-first nearest neighbor queries. We conclude in Section 7.

2 Background

R-trees [4] and their variants (e.g. [5, 6]. See [1] for a list of others) are data structures for organizing spatial objects in Euclidean space. They support dynamic insertion and deletion operations. Internal and leaf nodes contain records. A record r belonging to an internal node is a tuple $\langle M, \mu \rangle$ where μ is a pointer to the child node of r and M is an n-dimensional minimum bounding rectangle (MBR). M tightly bounds the spatial objects located in the subtree of r. For example, given the points $(1, 2)$, $(4, 5)$, and $(3, 7)$ in 2-space, the MBR would be $\langle (1, 4), (2, 7) \rangle$. The records of leaf nodes are also tuples but have the form $\langle M, o \rangle$ where o is either the actual spatial object or a reference to it.

The number of records in a node is its branching factor. Every node of an R-tree contains between b and B records where both b and B are positive integers and $b \leq \lfloor \frac{B}{2} \rfloor$. One exception is the root node which must have at least two records. R-trees are completely balanced—all leaf nodes have the same depth. Figure 1 depicts an example collection of spatial objects, their MBRs, and their R-tree. More details on R-trees are available in [1].

We consider nearest neighbor searches on R-trees. In all cases these searches have the form: *Given a query point q and an R-tree T with spatial objects of matching dimension to q, find the k-nearest objects to q in T.* Nearest here and throughout the rest of the paper is defined by *Euclidean distance.*

3 Nearest Neighbors

There are two dominant nearest neighbor algorithms for the R-tree. The first is a best-first search algorithm (denoted HS) due to Hjatlson and Samet [7]. HS is optimal in the sense that it only searches nodes with bounding boxes intersecting the k-nearest neighbor hypersphere [7, 8]. However, it has worst case space complexity that is linear in the total number of tree nodes. With large data sets this cost may become prohibitive [3].

The second algorithm due to Roussopoulos et al. [9] (denoted here by RKV) is a branch and bound depth-first search. RKV employs several heuristics to prune away branches of the tree. We define and discuss the

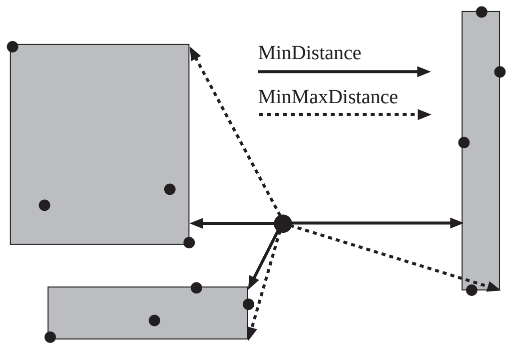

Fig. 2. A visual explanation of MinDist and MinMaxDist in two dimensions.

subtlety of these heuristics below. While RKV may search more nodes than HS, it has worst-case space complexity that is only logarithmic in the number of tree nodes. In addition, the authors of [10] note that statically constructed indices map all pages on a branch to contiguous regions on disk, so a depth-first search may "yield fewer disk head movements than the distance-driven search of the HS algorithm." In these cases RKV may be preferable to HS for performance reasons beyond space complexity.

3.1 Distances

RKV uses three different strategies (here called H1, H2, and H3 respectively and defined formally below) to prune branches. H3 is based on a measure called MinDist which gives the actual distance between a node and a query point. In other words, MinDist(q, M) is the length of the shortest line between the query point q and the nearest face of the MBR M. When the query point lies within the MBR, MinDist is 0. Figure 2 shows the MinDist values for a query point and three minimum bounding rectangles. Because an MBR tightly encapsulates the spatial objects within it, each face of the MBR must touch at least one of the objects it encloses [9]. This is called the MBR face property.

Figure 2 shows that MinDist is a lower bound or optimistic estimate of the distance between the query point and some spatial object inside its MBR. However, the actual distance between a query point and the closest object may be much larger.

H1 and H2 use a second measure called MinMaxDist which provides an upper bound on the distance between an actual object in a node and a query point. In other words, MinMaxDist provides a pessimistic estimate of the distance between the query point and some spatial object within its MBR. Figure 2 depicts these distances for a query point and three MBRs. From its name one can see MinMaxDist is calculated by finding the minimal distance from a set of maximal distances. This set of maximal distances is formed as follows: Suppose we have an n-dimensional minimum bounding rectangle. If we fix one of the dimensions, we are left with two $n - 1$ dimensional hyperplanes; one representing the MBR's lower bounds, the other representing its upper bounds. We know from the MBR face property that at least one spatial object touches each of these hyperplanes. However, given only the MBR, we cannot identify this location. But, given a query point, we can say that an object is at least as close as the distance from that point to the farthest point on the closest hyperplane. This distance is an upper bound on the distance between the query point and a spatial object located within the MBR. By iteratively fixing each dimension of an MBR and finding the upper bound, we can form the set of maximal distances. Since each maximal distance is an upper bound, it follows that the minimum of these is also an upper bound. This minimum distance is what we call the MinMaxDist(q, M) of a query point q and an MBR M.

3.2 Pruning Heuristics

Pruning strategies based on MinDist and MinMaxDist potentially remove large portions of the search space. The following three strategies were originally defined in [9] for use in RKV. All assume a query point q and a list of MBRs $\mathcal{M}$ (to potentially prune) sorted by MinDist. The latter assumption is based on

Algorithm 1 $1\text{NN}(q, n, e)$

Require: A query point q, a node n and a nearest neighbor estimate e. e may be a distance estimate object or a (pointer to a) spatial object.

```
1: if LeafNode(n) then
2:     for $\langle M, o \rangle$ in $records[n]$ do {$M$ is a MBR, $o$ is a (pointer to a) spatial object}
3:         if $\text{Dist}(q, M) \leq \text{Dist}(q, e)$ then
4:             $e \leftarrow o$
5:         end if
6:     end for
7: else
8:     $ABL \leftarrow \text{Sort}(\text{records}[n])$ {Sort records by MinDist }
9:     for $\langle M, \mu \rangle$ in $ABL$ do {$M$ is an MBR, $\mu$ points to a child node}
10:        if MinMaxDist$(q, M) \leq e$ then {H2* Pruning}
11:            $e \leftarrow \text{MinMaxDist}(q, M)))$
12:        end if
13:    end for
14:    for $\langle M, \mu \rangle$ in $ABL$ do {$M$ is an MBR, $\mu$ points to a child node}
15:        if $\text{MinDist}(q, M) < \text{Dist}(q, e)$ then {H3 Pruning}
16:            $1\text{NN}(q, \mu, e)$
17:        end if
18:    end for
19: end if
```

empirical results from both [9] and [7]. In addition, two strategies, H2 and H3, assume a current nearest object o.

Definition 1 (H1). *Discard any MBR* $M_i \in \mathcal{M}$ *if there exists* $M_j \in \mathcal{M}$ *with* $\text{MinDist}(q, M_i) > \text{MinMaxDist}(q, M_j)$

Definition 2 (H2). *Discard* o *if there exists* $M_i \in \mathcal{M}$ *such that* $\text{MinMaxDist}(q, M_i) < \text{Dist}(q, o)$.

Definition 3 (H3). *Discard any minimum bounding rectangle* $M \in \mathcal{M}$ *if* $\text{MinDist}(q, M) > \text{Dist}(q, o)$

Both Cheung et al. [11] and Hjaltason et al. [7] show that any node pruned by H1 is also pruned by H3. Furthermore, they note that H2 serves little purpose since it does not perform any pruning. This has led to the development of simpler but behaviorally-identical versions of RKV that rely exclusively on H3 for pruning. As a result, *we take* RKV *to mean the original* RKV *without H1 and H2.*

The benefits of MinMaxDist, however, should not be overlooked — it can provide very useful information about unexplored areas of the tree. The key is to replace an actual object o with a distance estimate e (some call this the closest point candidate) and then adjust H2 so that we *replace* the estimate with the MinMaxDist instead of *discarding* the object. This gives us a new definition of H2 which we call H2*.

Definition 4 (H2*). *Replace* e *with* $\text{MinMaxDist}(q, M_i)$ *if there exist* $M_i \in \mathcal{M}$ *such that* $\text{MinMaxDist}(q, M_i) < e$.

This definition is not new. In fact, the authors of [2] use *replace* instead of *discard* in their description of H2. However, updating the definition of H2 to H2* in RKV does not yield the full pruning power of MinMaxDist. We need to apply H2* early in the search process. This variation on RKV yields Algorithm 1 which we refer to it as 1NN. Note that the RKV traditionally applies H2 after line 16. This diminishes the power of pessimistic pruning. In fact, our exponential speedup results in Sections 4 and 5 hold even when H2* replaces H2 in the original algorithm.

3.3 k-Nearest Neighbors

Correctly generalizing H2* to k-nearest neighbor queries is essential in light of the potential power of pessimistic pruning. However, as Böhm et. al [10] point out, such an extension takes some care.

We begin by replacing e with a priority queue L of k-closest neighbors estimates. Note that H2* doesn't perform direct pruning, but instead, updates the neighbor estimate when distance guarantees can be made. If the MinMaxDist of a node is less than the current distance estimate, then we can update the estimate because the future descent into that node is guaranteed to contain an object with actual distance at most MinMaxDist. We call estimates in updates of this form *promises* because they are not actual distances, but are upper bounds on distances. Moreover each estimate is a promise of, or place holder for, a spatial object that is as least as good the promise's prediction. A natural but incorrect generalization of H2 places a promise in the priority queue whenever the maximum-distance element in L is farther away than the MinMaxDist. This leads to two problems. First, multiple promises may end up referring to the same spatial object. Second, a promise may persist past its time and eventually refer to a spatial object already in the queue. These problems are depicted visually in Figure 3. The key to avoiding both problems is to always remove a promise from the queue before searching the node which generated it; it will always be replaced by an equal or better estimate or by an actual object. This leads us to the following generalization of H2 which we call Promise-Pruning:

Definition 5 (Promise-Pruning). *If there exists $M_i \in \mathcal{M}$ such that $\delta(q, M_i) = \text{MinMaxDist}(q, M_i) < \mathit{Max}(L)$, then add a promise with distance $\delta(q, M_i)$ to L. Additionally, replace any promise with distance $\delta(q, M_i)$ from L with ∞ before searching M_i.*

This generalization is tantamount to an extension suggested by Böhm et al. in [10]. Our primary contribution is to show that this extension, when performed at the right point, may provide an exponential performance speedup on depth-first nearest neighbor queries.

For completeness, we also provide generalizations of H1 and H3 which we call K1 and K3 respectively. We also prove that K3 dominates K1, just as it did in the 1-nearest neighbor case. This proof is a simple extension of those appearing in [11, 7].

Definition 6 (K1). *Discard any minimum bounding rectangle $M_i \in \mathcal{M}$ if there exists $\mathcal{M}' \subseteq \mathcal{M}$ such that $|\mathcal{M}'| \geq k$ and for every $M_j \in \mathcal{M}'$ it is the case that $\text{MinDist}(q, M_i) > \text{MinMaxDist}(q, M_j)$*

Definition 7 (K3). *Discard any minimum bounding rectangle $M_i \in \mathcal{M}$ if $\text{MinDist}(q, M_i) > \mathit{Max}(L)$ where Max returns the largest estimate in the priority queue.*

Theorem 1. *Given a query point q, a list of MBRs $\mathcal{M}$, and a priority queue of k closest neighbor estimates L, any MBR pruned by K1 in the depth-first RKV algorithm is also pruned by K3.*

Proof. Suppose we are performing a k nearest neighbor search with query point q and K1 prunes MBR M from $\mathcal{M}$. From Definition 6 there exists $\mathcal{M}' \subset \mathcal{M}$ such that $|\mathcal{M}'| \geq k$ and every M' in $\mathcal{M}'$ has $\text{MinMaxDist}(q, M') < \text{MinDist}(q, M)$. Since for any MBR N, $\text{MinDist}(q, N) \leq \text{MinMaxDist}(q, N)$, each M' in $\mathcal{M}'$ will be searched before M because $\mathcal{M}$ is sorted by MinDist. Because each M' in $\mathcal{M}'$ is guaranteed to contain a spatial object with actual distance at most that of than an object found in M and since we have $|\mathcal{M}'| \geq k$ we know $\text{Max}(L) < \delta(q, M)$. Therefore, from Definition 7, M would also be pruned using K3.

Theorem 1 means K1 is redundant with respect to K3, so we do not use it in our depth-first k-nearest neighbor procedure outlined in Algorithm 2. We call this procedure KNN.

4 The Power of Pessimism

In this section we show that 1NN may perform exponentially faster than RKV despite the fact that it differs only slightly from the original definition. As we noted earlier, the original RKV is equivalent to RKV with only H3 pruning. Thus, RKV is Algorithm 1 without lines 9-13.

Theorem 2. *There exists a family of R-tree and query point pairs $\mathcal{T} = \{(T_1, q_n), \ldots, (T_m, q_m)\}$ such that for any (T_i, q_i), RKV examines $O(n)$ nodes and 1NN examines $O(\log n)$ nodes on a 1-nearest neighbor query.*

Algorithm 2 KNN(k, q, n, L)

Require: An integer k, a query point q, a node n and a priority queue L of fixed size k. L initially contains k neighbor estimates with distance ∞.

```
1: if LeafNode(n) then
2:     for ⟨M, o⟩ in records[n] do {M is a MBR, o is a (pointer to a) spatial object}
3:         if Dist(q, M) < max(L) then {Here max returns the object or estimate of greatest distance}
4:             insert(L, o) {Inserting o into L replaces some other estimate or object.}
5:         end if
6:     end for
7: else
8:     ABL ← Sort(records[n]) {Sort records by MinDist }
9:     for ⟨M, μ⟩ in ABL do {M is an MBR, μ points to a child node}
10:        if MinMaxDist(q, M) < max(L) then {Promise-Pruning}
11:            insert(L, Promise(MinMaxDist(q, M)))
12:        end if
13:    end for
14:    for ⟨M, μ⟩ in ABL do {M is an MBR, μ points to a child node}
15:        if MinDist(q, M) < max(L) then {K3 Pruning}
16:            if L contains a promise generated from M then
17:                remove(L, Promise(MinMaxDist(q, M)))
18:            end if
19:            KNN(k, q, μ, L)
20:        end if
21:    end for
22: end if
```

Proof. For simplicity, we restrict our attention to R-trees composed of points in $\mathbb{R}^2$ so that all the MBRs are rectangles. Also, we only construct complete binary trees where each node has two records. The construction follows the illustration in Figure 4. Let $\delta(i, j)$ be the Euclidean distance from point i to point j and let (i, j) be their rectangle. Let q be the query point. Choose three points a, b, and c such that $\delta(q, a) = r_1 < \delta(q, b) = r_4 < \delta(q, c)$ and (b, c) forms a rectangle W with a corner a. Similarly, choose three points d, e, and f such $r_1 < \delta(q, f) = r_2 < \delta(q, d) = r_3 < r_4 < \delta(q, e)$ and (d, e) forms a rectangle X with corner f. Let T be a complete binary tree over n leaves where each node has two records. Let T_1 be the left child of T and let T_2 be the right child of T. Let L_1 be the far left leaf of T_1 and let L_2 be the far left leaf of T_2. Place b and c in L_1, and d and e in L_2. In the remaining leaves of T_1, place pairs of points (p_i, p_j) such that p_i and p_j are interior to V, $\delta(q, p_i) > r_4$ and $\delta(q, p_j) > r_4$ but (p_i, p_j) form a rectangle with corner p' such that $r_3 < \delta(q, p') < r_4$. Rectangles Y and Z in Figure 4 are examples of this family of point pairs. In the remaining leaves of T_2 place pairs of points (p_k, p_l) such that p_k and p_l are interior to U (*i.e.*, so that MinDist$(q, (p_k, p_l)) > r_3$). The construction yields a valid R-tree because we place pairs of points at the leaves and build up the MBRs of the internal nodes accordingly.

Claim. Given a tree T and query point q as constructed above, both RKV and 1NN prune away all of T_2 save the left branch down to L_2 on a 1-nearest neighbor query.

Proof. Note that d is the nearest neighbor to q in T so both algorithms will search T_2. L_2, by construction, has the least MinDist of any subset of points in T_2, so both algorithms, when initially searching T_2, will descend to it first. Since $\delta(q, d)$ is the realization of this MinDist and no other pair of points in X has MinDist $< \delta(q, d)$, both algorithms will prune away the remaining nodes using H3.

Now we'll show that RKV must examine all the nodes in T_1 while 1NN can use information from X to prune away all of T_1 save the left branch down to L_1.

Lemma 1. *Given an R-tree T and query point q as constructed above, RKV examines every node in T_1 on a 1-nearest neighbor query.*

Proof. Since MinDist$(q, W) = \delta(q, a)$, RKV descends to L_1 first and claims b as its nearest neighbor. However, RKV is unable to prune away any of the remaining leaves of T_1. To see this, let $L_i = (p_{i1}, p_{i2})$ and $L_j =$

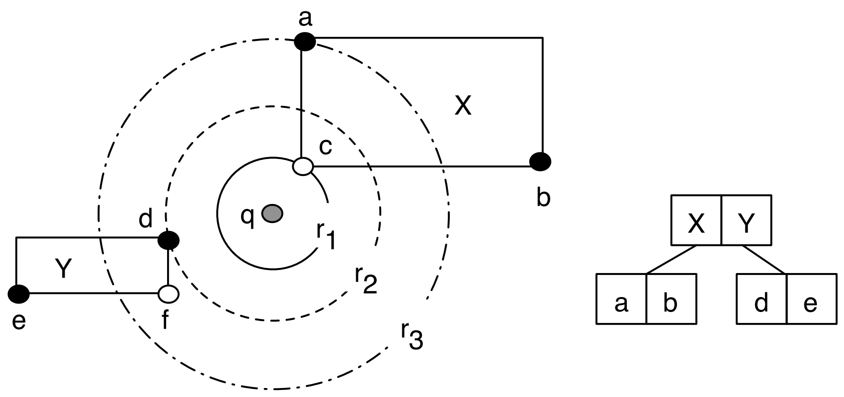

Fig. 3. Blindly inserting promises into the queue, without removing them correctly, wreaks havoc on the results. For example, when performing a 2-nearest neighbor search on the tree above, a promise with distance f is placed in the queue at the root. After investigating X, the queue retains f and a. However, if f is not removed before descending into Y, the final distances in the queue are d and f — an incorrect result.

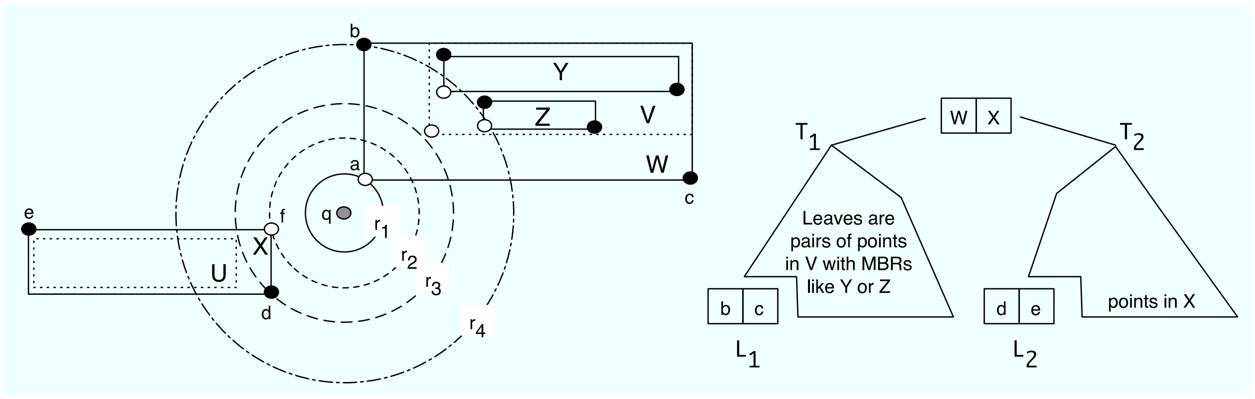

Fig. 4. A visual explanation of the tree construction for Theorem 2

(p_{j1}, p_{j2}) be distinct leaves of T_1 (but not L_1). Note that $\textsc{MinDist}(q,(p_{i1},p_{i2})) < r_4 < \min(\delta(q,p_{j1}),\delta(q,p_{j2}))$ and $\textsc{MinDist}(q,(p_{j1},p_{j2})) < r_4 < \min(\delta(q,p_{i1}),\delta(q,p_{i2}))$. This means that the MinDist of any leaf node is at most r_4 but every point is at least r_4 so RKV must probe every leaf. As a result, it cannot prune away any branches.

Lemma 2. *Given a tree T and query point q as constructed above,* 1NN *prunes away all nodes in T_1 except those on the branch leading to L_1 in a 1-nearest neighbor query.*

Proof. 1NN uses the MinMaxDist information from X as an indirect means of pruning. Before descending into T_1, the algorithm updates its neighbor estimate with $\delta(q,d)$. Like RKV, 1NN descends into T_1 directly down to L_1 since $\delta(q,W) < \delta(q,d)$ and W has the smallest MinDist of all the nodes. Unlike RKV, it reaches and ignores b because $\delta(q,d) < \delta(q,b)$. In fact, the promise granted by X allows us to prune away all other branches of the tree since the remaining nodes are all interior to V and $\delta(q,d) < \textsc{MinDist}(q,V)$.

The theorem follows from Lemma 1 and Lemma 2. RKV searches all of T_1 ($O(n)$ nodes) while 1NN searches only the paths leading to L_1 and L_2 ($O(\log n)$ nodes). As a consequence, 1NN can reduce the search space exponentially over RKV. □

In the original RKV, H2 pruning appears on line 16. Our results hold even if we replace H2 with H2*. This is because all the pruning in T_1 relies on the MinMaxDist found at the root node. Hence the promotion

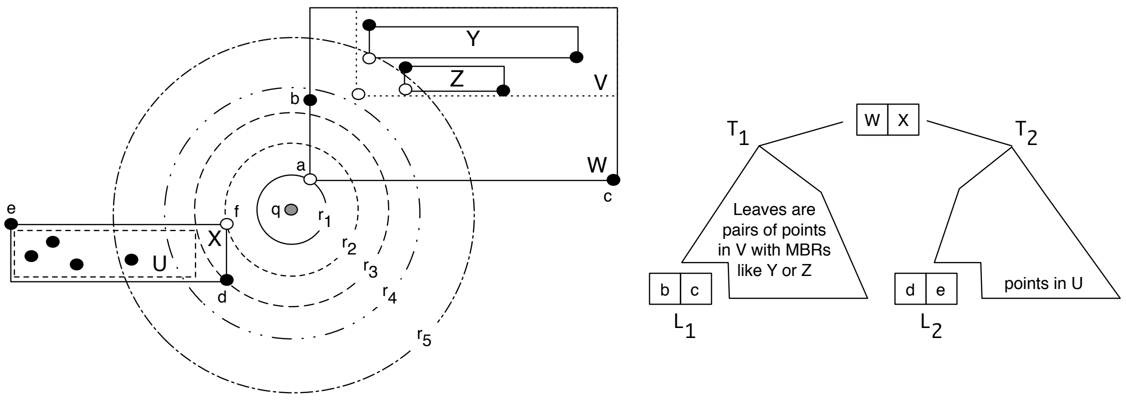

Fig. 5. A visual explanation of the tree construction used in Theorem 3.

of pessimistic pruning in Algorithm 1 plays a crucial role in the performance of depth-first nearest neighbor queries.

5 Search Space Reductions with K-Nearest Neighbors

Here we show that the benefits 1NN reaps from MinMaxDist extend to KNN when H2* is properly generalized to Promise-Pruning. In particular, we construct a class of R-trees where KNN reduces the number of nodes visited exponentially when compared with RKV. As in Section 4, we take RKV to mean Algorithm 2 without lines 9-13 (and additionally lines 16-18).

Theorem 3. *There exists a family of R-trees and query point pairs $\mathcal{T} = \{(T_1, q_n), \ldots, (T_m, q_m)\}$ such that for any (T_i, q_i), RKV examines $O(n)$ nodes and KNN examines $O(\log n)$ nodes on a 2-nearest neighbor query.*

Proof. The proof follows the outline of Theorem 2. The construction is similar to Figure 4 except that b shifts slightly down and V shifts slightly up so that $\delta(q, b) < \text{MinDist}(q, V)$. We illustrate this in Figure 5 and give details where the two proofs differ.

Claim. Given a tree T and query point q as constructed in Figure 5, both KNN and RKV prune away all of T_2 save the left branch down to L_2 on a 2-nearest neighbor query.

Proof. Note that b and d are the two nearest-neighbors to q in T. Both algorithms will search T_1 first because $\text{MinDist}(q, W) < \text{MinDist}(q, X)$. Since both algorithms are depth-first searches, b is in tow by the time T_2 is searched. Because d has yet to be realized, both algorithms will search T_2 and prune away the remaining nodes just as in 4.

Just as before, we'll show that RKV must examine all the nodes in T_1 while KNN can use information from X to prune away all of T_1 save the left branch down to L_1.

Lemma 3. *Given an R-tree T and query point q as constructed above, RKV examines every node in T_1 in a 2-nearest neighbor search.*

Proof. Since $\text{MinDist}(q, W) = \delta(q, a)$, RKV descends to L_1 first and inserts b (and c) into its 2-best priority queue. However, RKV is unable to prune away any of the remaining leaves of T_1 because every pair of leaf points have MinDist at most r_5 but all points in T_1 (besides b) lie outside r_5 . As a result RKV must probe every leaf.

Lemma 4. *Given a tree T and query point q as constructed above, KNN prunes away all nodes in T_1 except those on the branch leading to L_1 in a 2-nearest neighbor search.*

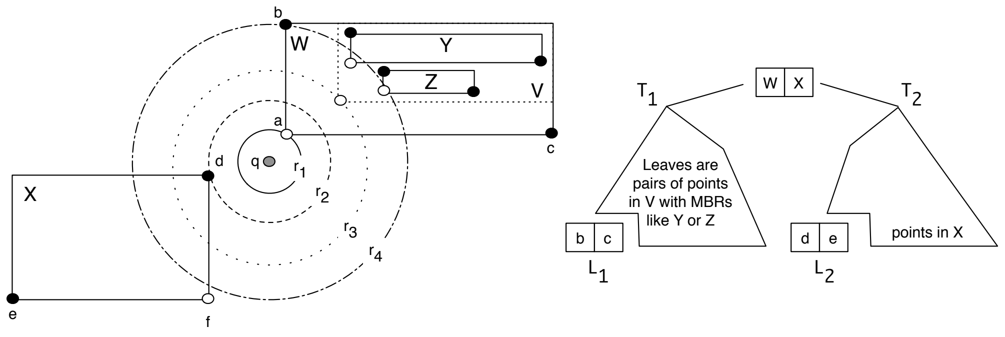

Fig. 6. A visual explanation of the tree construction used in Theorem 4.

Proof. Before descending into T_1, KNN inserts a promise with distance $\text{MinMaxDist}(q, X) = \delta(q, d)$ into its 2-best priority queue. The algorithm descends into T_1 directly down to L_1, finding b and inserting it into its 2-best priority queue. Unlike RKV, the promise granted by X allows us to prune away all other nodes of the tree since the remaining nodes are all interior to V and $\delta(q, d) < \text{MinDist}(q, V)$.

The theorem follows from Lemma 3 and Lemma 4. Note that if KNN did not remove the promise granted by d at X the final result would be the point d and its promise – an error. □

We can generalize the construction given in Theorem 3 so that the exponential search space reduction holds for any k nearest neighbor query. In particular, for any constant $k > 0$ there exists a class of R-tree and query point pairs for which KNN reduces the search space exponentially over RKV. A simple way of accomplishing this is to place $k - 1$ points at b and insert them into the far left leaves of T_1.

6 Time / Space Tradeoffs

Given the search space reductions offered by 1NN and KNN, it is natural to ask if the space-efficient depth-first algorithms can approach the time efficiency of the best-first algorithms. In other words, how does KNN stack up against HS? We answer this question here in the negative by constructing a family of R-trees where HS performs exponentially better than KNN. The HS algorithm uses a priority queue to order the nodes by MinDist. It then performs a best-first search by MinDist while pruning away nodes using K3. We direct the reader to [7] for more details.

Theorem 4. *There exists a family of R-tree and query point pairs* $\mathcal{T} = \{(T_1, q_n), \ldots, (T_m, q_m)\}$ *such that for any* (T_i, q_i)*, HS examines* $O(\log n)$ *nodes and* 1NN *examines* $O(n)$ *nodes on a 1-nearest neighbor query.*

Proof. This construction resembles the construction in Theorem 2 and is depicted visually in Figure 6. We organize T_1 in exactly the same way as Theorem 2: we choose three points a, b, and c such that $\delta(q, a) = r_1 < \delta(q, b) = r_4 < \delta(q, c)$ and (b, c) forms a rectangle W with a corner a. Next we choose three points d, e, and f such $r_1 < \delta(q, d) = r_2 < r_4 < \delta(q, f) < \delta(q, e)$ and (d, e) forms a rectangle X with corner f. Let T be a complete binary tree over n leaves where each node has two records. Let $T_1, T_2, L_1,$ and L_2 be as in Theorem 2. Place b and c in L_1, and d and e in L_2. In the remaining leaves of T_1, place pairs of points (p_i, p_j) such that p_i and p_j are interior to V, $\delta(q, p_i) > r_4$ and $\delta(q, p_j) > r_4$ but (p_i, p_j) form a rectangle with corner p' such that $r_3 < \delta(q, p') < r_4$. Rectangles Y and Z in Figure 6 are examples of this family of point pairs. In the remaining leaves of T_2 place pairs of points (p_k, p_l) such that p_k and p_l are interior to X.

Claim. Given a tree T and query point q as constructed above, both 1NN and HS prune away all of T_2 save the left branch down to L_2 on a 1-nearest neighbor query.

Proof. d is the nearest neighbor to q in T so both algorithms must search T_2. L_2, by construction, has the least MinDist of any subset of points in T_2, so both algorithms, when initially searching T_2, will descend to it first. Since $\delta(q, d)$ is the realization of this MinDist and no other pair of points in X has $\text{MinDist} < \delta(q, d)$, both algorithms will prune away the remaining nodes using H3.

Now we'll show that 1NN must examine all the nodes in T_1 while HS can use the MinDist of X to bypass searching all of T_1 save the path down to L_1.

Lemma 5. *Given an R-tree T and query point q as constructed above,* 1NN *examines every node in T_1 on a 1-nearest neighbor query.*

Proof. 1NN descends into T_1 before T_2 since $\textsc{MinDist}(q, W) < \textsc{MinDist}(q, X)$. Note that the distance estimate delivered by f is useless given that every pair of leaf points in T_1 has MinDist less than $\delta(q, f)$. 1NN will descend to L_1 and find b but it cannot rule out the rest of T_1 because every pair of leaf points forms a rectangle with MinDist smaller than r_4. To see this, let $L_i = (p_{i1}, p_{i2})$ and $L_j = (p_{j1}, p_{j2})$ be distinct leaves of T_1 (but not L_1). Note that $\textsc{MinDist}(q, (p_{i1}, p_{i2})) < r_4 < \min(\delta(q, p_{j1}), \delta(q, p_{j2}))$ and $\textsc{MinDist}(q, (p_{j1}, p_{j2})) < r_4 < \min(\delta(q, p_{i1}), \delta(q, p_{i2}))$. This means that the MinDist of any leaf node is at most r_4 but every point is beyond r_4 so 1NN cannot use H3 pruning. Furthermore, since MinMaxDist is always an upper bound on the points, it can never use H2* pruning. Thus, 1NN searches all of T_1.

Lemma 6. *Given a tree T and query point q as constructed above, HS prunes away all nodes in T_1 except those on the branch leading to L_1 in a 1-nearest neighbor query.*

Proof. Like 1NN, HS descends directly to L_1, however, once b is in tow, it immediately jumps back to X since $\textsc{MinDist}(q, X) < \textsc{MinDist}(q, V)$. Since d is the 1-nearest neighbor and since $\textsc{MinDist}(q, X) = \delta(q, d)$, it immediately descends to L_2 to find d. Since $\delta(q, d) < \textsc{MinDist}(q, V)$ it can use H3 to prune away the rest of T_1.

The theorem follows from Lemma 5 and Lemma 6. 1NN searches all of T_1 ($O(n)$ nodes) while HS searches only the paths leading to L_1 and L_2 ($O(\log n)$ nodes). As a consequence, HS can prune the search space exponentially over 1NN even when 1NN has the advantages of H2*. □

Extending Theorem 4 to k-nearest neighbors is fairly straight-forward. Add $k - 1$ points to X between r_2 and r_3 and place these points in leaves as adjacent to L_2 as possible. Since this set of points forms a rectangle with MinDist smaller than r_3 and since this rectangle is encountered en route to L_2, HS will find the points immediately and then use H3 to prune away the rest of T_1 and T_2. This gives us the following theorem:

Theorem 5. *There exists a family of R-tree and query point pairs $\mathcal{T} = \{(T_1, q_n), \ldots, (T_m, q_m)\}$ such that for any (T_i, q_i), HS examines $O(\log n)$ nodes and* KNN *examines $O(n)$ nodes on a k-nearest neighbor query.*

7 Open Problems and Future Work

The most natural open problem is quantifying the time/space trade-off of depth-first versus best-first nearest-neighbor algorithms on the R-tree. One line of future work might explore hybrid algorithms that combine the space-efficiency of depth-first search along with the time-efficiency of best-first search.

References

1. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-Trees: Theory and Applications. 1 edn. Advanced Information and Knowledge Processing. Springer-Verlag, London (2006)
2. Papadopoulos, A., Manolopoulos, Y.: Performance of nearest neighbor queries in r-trees. In: Proceedings of the 6th International Conference on Database Theory. (1997) 394–408
3. Berchtold, S., Böhm, C., Keim, D.A., Krebs, F., Kriegel, H.P.: On optimizing nearest neighbor queries in high-dimensional data spaces. In: ICDT. (2001) 435–449
4. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1984) 47–57
5. Sellis, T., Roussopoulos, N., Faloutsos, C.: R+-tree: A dynamic index for multidimensional objects. In: Proceedings of the 13th International Conference on Very Large Databases. (1988) 507–518
6. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: R*-tree: An efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1990) 322–331

7. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Transactions on Database Systems **24** (1999) 265–318
8. 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 Sixteenth ACM Symposium on Principles of Database Systems, ACM Press (1997) 78–86
9. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proceedings ACM SIGMOD Internaiontal Conference on the Management of Data. (1995) 71–79
10. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys (CSUR) **33** (2001) 322–373
11. Cheung, K.L., Fu, A.W.C.: Enhanced nearest neighbour search on the r-tree. SIGMOD Record **27** (1998) 16–21