Balanced Aspect Ratio Trees Revisited

Amitabh Chaudhary[1] and Michael T. Goodrich[2]

[1] Department of Computer Science & Engineering,
University of Notre Dame, Notre Dame IN 46556, USA
Amitabh.Chaudhary.1@nd.edu
[2] Department of Computer Science,
Bren School of Information & Computer Sciences,
University of California, Irvine CA 92697, USA
goodrich@acm.org

Abstract. Spatial databases support a variety of geometric queries on point data such as range searches, nearest neighbor searches, etc. Balanced Aspect Ratio (BAR) trees are hierarchical space decomposition structures that are general-purpose and space-efficient, and, in addition, enjoy a worst case performance poly-logarithmic in the number of points for approximate queries. They maintain limits on their depth, as well as on the aspect ratio (intuitively, how skinny the regions can be). BAR trees were initially developed for 2 dimensional spaces and a fixed set of partitioning planes, and then extended to d dimensional spaces and more general partitioning planes. Here we revisit 2 dimensional spaces and show that, for any given set of 3 partitioning planes, it is not only possible to construct such trees, it is also possible to derive a simple closed-form upper bound on the aspect ratio. This bound, and the resulting algorithm, are much simpler than what is known for general BAR trees. We call the resulting BAR trees Parameterized BAR trees and empirically evaluate them for different partitioning planes. Our experiments show that our theoretical bound converges to the empirically obtained values in the lower ranges, and also make a case for using evenly oriented partitioning planes.

1 Introduction

Spatial databases for scientific applications need efficient data structures to solve a variety of geometric queries. Consider, e.g., the Sloan Digital Sky Survey (SDSS) [16, 17], a scientific application with which we have direct experience. It stores light intensities for over a 100 million celestial objects as points on a two-dimensional sphere, and needs support for geometric queries like the nearest neighbor queries, proximity queries, and general range queries (not just axis orthogonal). Similar needs arise in geographical information systems.

There are many access methods based on the hierarchical space decomposition data structures that are useful in solving geometric queries on point data. Quad trees and k-d trees are widely popular examples (e.g., see Samet [18, 19]). In these data structures two properties, *depth* and *aspect ratio*, play a crucial role

F. Dehne, A. López-Ortiz, and J.-R. Sack (Eds.): WADS 2005, LNCS 3608, pp. 73–85, 2005.

in determining their efficiency in solving queries. The depth of a tree characterizes the number of nodes that have to be visited to find regions of interest. The aspect ratio of a tree (intuitively, how "skinny" a region can be) characterizes the number of wasteful nodes that can be in any particular region of interest. Most queries require both of these values to be small for them to be solved efficiently.

Unfortunately, both quad trees and k-d trees optimize one of these properties at the expense of the other. Quad trees produce regions with optimal aspect ratios, but they can have terrible depth. K-d trees, on the other hand, have optimal (logarithmic) depth, but they often produce lots of long-and-skinny regions, which slow query times.

Balanced Aspect Ratio (BAR) trees are hierarchical space decomposition data structures that have logarithmic depth and bounded aspect ratios. As a result they have a worst case performance poly-logarithmic in the number of points for approximate queries such as the approximate nearest neighbor, approximate farthest neighbor, approximate range query, etc. They were initially developed for 2 dimensional spaces [12] for a particular fixed set $(0, \pi/4, \pi/2)$ of partitioning planes. Then [10], they were extended to d dimensional spaces, and the partitioning planes could be chosen flexibly as long as certain conditions were met. These conditions when applied to small dimensional spaces give bounds that are known to be very loose. For instance, in d dimensional spaces, as long as d of the partitioning planes are axis orthogonal, the aspect ratio (we give a precise definition later) is bounded by $50\sqrt{d}+55$. In 2 dimensional spaces, the aspect ratio is bounded by a number very close to 6. In this paper, we are interested in developing simpler conditions for choosing partitioning planes flexibly in BAR trees for small dimensions. This will allow us to derive tighter bounds for the aspect ratio, and discover the best set of partitioning planes for a particular application.

1.1 Related Prior Work

In this subsection, we briefly review some known general-purpose hierarchical spatial decomposition trees for a set of points, S.

The Binary Space Partitioning (BSP) Trees. The BSP tree [14, 13] is a recursive subdivision of space into regions by means of half-planes. Each node u in the tree represents a convex region and the points from S lying in it. Initially, the root node r of the tree represents a bounding region of the point set S. At each node u, an associated line partitions the region of u, R_u, into two disjoint regions R_l and R_r. The node u then has two child nodes l and r representing R_l and R_r respectively. If the number of points from S in R_u is less than some constant, u is not partitioned and becomes a leaf. These structures satisfy our condition of being general-purpose, but without further restricting how the cutting lines are chosen, these structures are inefficient. Thus, much work has been done on methods for specializing BSP trees to be more efficient, which we review next.hfil

The k-d Tree. This structure was introduced by Bentley [3, 4, 5] and has been extensively studied. It is a special class of the BSP tree: the partitioning line

is orthogonal to one of the axes and such that it divides the set of points at the node in half by cardinality. This guarantees that the depth of the tree is $O(\log n)$. So point location queries, which take time proportional to the depth, can be answered efficiently. But, since there are no guarantees on the aspect ratio of the regions produced, with some exceptions [9], the running times of queries can nevertheless be poor.

The Quadtree. The quadtree (e.g., see [18, 19]) is another special class of the BSP tree. The point set S is initially bounded by a square, and the partitions are such that a square region is divided into four smaller squares of equal area. (This notion can be extended to d dimensional space, giving rise to a structure called the *octree*.) The aspect ratios of regions in quadtrees is bounded by a constant, but these trees can have unbounded depth. So even basic point location queries can take unbounded time. If the point set is uniformly distributed, however, then the depth is bounded, and in those situations quadtrees perform well for some geometric queries.

Balanced Box Decomposition Trees. In [1, 2], Arya *et al.* describe a relative of a binary-space partitioning tree called the *Balanced Box Decomposition* (BBD) tree. This structure is based on the fair-split tree of Callahan and Kosaraju [6, 7] and is defined such that its depth is $O(\log n)$ and all of the associated regions have low combinatorial complexity and bounded aspect ratio. Arya *et al.* show how BBD trees guarantee excellent performance in approximating general range queries and nearest-neighbor queries. (Approximate queries are like the regular exact versions, except they allow an error. See [10] for formal definitions.) The aspect ratio bound on each region allows them to bound the number of nodes visited during various approximate query searches by limiting the number of nodes that can be packed inside a query region. However, since these trees rely on using hole cuts during construction, they produce non-convex regions and thus are not true BSP trees. This is also a drawback with respect to several applications in computer graphics and graph drawing, where convexity of the partitioned regions is desirable (e.g., see [12, 15]).

Balanced Aspect Ratio Trees. Duncan *et al.* [12, 11] introduced the *Balanced Aspect Ratio* (BAR) trees. These are similar to k-d trees in 2-dimensional space, except that instead of allowing only axis-orthogonal partitions, they also allow a third partition orthogonal to a vector at a $\pi/4$ angle to the axes. This extra cut allows them to find partitions that not only divide the point set in a region evenly, but also ensure that the child regions have good (bounded) aspect ratio. In [10], Duncan extended BAR trees to d dimensions. He showed that if a certain set of conditions is satisfied, BAR trees with bounded aspect ratio can be constructed. He also proved bounds on the running time of approximate queries. The $(1+\varepsilon)$-nearest neighbor query and the $(1-\varepsilon)$-farthest neighbor query can be answered in $O(\log n + (1/\varepsilon)\log(1/\varepsilon))$ time, where n is the number of points. The ε-range query and the ε-proximity query can be answered in $O(\log n + (1/\varepsilon) + k)$ time, where k is the size of the output.

1.2 Our Contributions

In this paper, we introduce the Parameterized Balanced Aspect Ratio (PBAR) trees in 2 dimensions, which take any three vectors as the partitioning planes. They enjoy all the advantages of BAR trees: general pupose, space efficient, logarithmic depth, bounded aspect ratio, poly-logarithmic worst case bounds for approximate versions of spatial queries. In addition, they have a bound on the aspect ratio which is a simple closed-form function of the given partitioning planes. The proofs used are significantly different from those for earlier BAR trees: they use the advantages of 2 dimensional spaces and yet work of any given set of partitioning vectors.

Our motivation for introducing PBAR trees comes from our experience with the Sloan Digital Sky Survey (SDSS) [16, 17]. Because objects in the SDSS are indexed by their positions on the night sky, the data can be viewed at a first level of indexing as two-dimensional points on a sphere. To allow for efficient access, astronomers overlay a quasi-uniform triangular "grid" on this sphere, to reduce the curvature of each "leaf" triangle to be "almost" planar and to reduce the number of points in each such triangle to a few hundred thousand. The difficulty is that when these leaf triangles are mapped to a projection plane to allow for fast queries via a secondary data structure, there are many different side angles that must be dealt with. A data structure like PBAR trees can conveniently use the given angles of a bounding triangle as its possible partitioning directions. Without this convenience, we would get poorly-shaped regions near the boundaries of these triangles.

The bounds on the running times for approximate queries, in [10], depend only on the fact that both BBD and BAR trees have $O(\log n)$ depth and ensure a constant bound on the aspect ratio of all their regions. These two conditions are satisfied by PBAR trees as well. So the same bounds hold for PBAR trees as well. We present empirical results for the $(1 + \varepsilon)$-nearest neighbor query using PBAR trees with various different partitioning planes using artificial data as well as real datasets from the SDSS. Our experiments also indicate that our bound is tight in some respects.

In the next section we give the foundations and definition for PBAR trees. In Section 3 we describe the algorithm for constructing PBAR trees and prove its correctness. In the last section we present our empirical results. We include details for the pseudo-code and proofs of correctness in an optional appendix in this extended abstract.

2 Parameterizing BAR Trees

PBAR trees are for point data in 2-dimensional space. The *distance* $\delta(p, q)$ between two points $p = (p_1, p_2)$ and $q = (q_1, q_2)$ is $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$. Extending this notion, the distance between two sets of points P and Q is

$$\delta(P, Q) = \min_{p \in P, q \in Q} \delta(p, q).$$

S is the set of n points given as input. The size $|R|$ of region R is the number of points from S in R.

Partitioning Vectors. We use vectors from $\mathbb{R}^2$ to specify partitioning directions. Note that partitioning vectors l and $-l$ are equivalent. The *angle* θ_{lm} between two partitioning vectors l and m is the angle from l to m in the counterclockwise direction, except that we take into account that m and $-m$ are equivalent. In the context of trigonometric functions, we shall prefer to use, for example, the short $\sin(lm)$ instead of $\sin(\theta_{lm})$.

To construct a PBAR tree, we use the 3 partitioning vectors from the given set $V = \{\lambda, \mu, \nu\}$. We make all partitions by taking a region R and dividing it into two subregions, R_1 and R_2, with a line c'_l, called a *cut*, orthogonal to some $l \in V$. A cut orthogonal to l is also called an *l-cut*. Note that if R is convex, both R_1 and R_2 are convex too. We divide the set of points in R, call it S, between R_1 and R_2 in the natural fashion. For points in S that are on c_l, we assign each of them to either R_1 or R_2 as per convenience.

Let the sequence (λ, μ, ν) be in the counterclockwise order. All the 3 sequences in the set $\mathcal{P}(V) = \{(\lambda, \mu, \nu), (\mu, \nu, \lambda), (\nu, \lambda, \mu)\}$ are equivalent for our purpose. So, often, we shall speak in terms of the general (l, m, n), where $(l, m, n) \in \mathcal{P}(V)$.

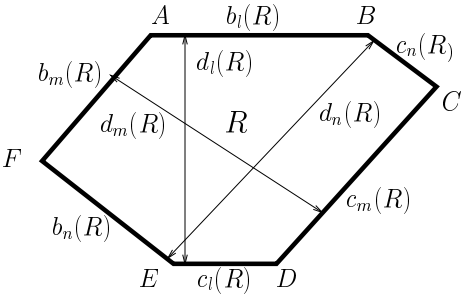

Fig. 1. The names used for the sides and the diameters

Canonical Regions and Canonical Aspect Ratios. We assume that the given set of points S has an initial convex bounding region with sides that are orthogonal to λ, μ, or ν. Since in constructing PBAR trees we make all partitions with lines orthogonal to these 3 partitioning vectors, the regions we construct are always hexagons with sides orthogonal to λ, μ, or ν. Some sides may be degenerate, that is, of length 0. We call these hexagonal regions *canonical regions*. See Figure 1. In a canonical region R, $b_l(R)$ and $c_l(R)$ are the two unique opposing sides orthogonal to the partitioning vector l. The *diameter* $d_l(R)$ of R with respect to the partitioning vector l is the distance $\delta(b_l(R), c_l(R))$. The *maximum diameter* of R is $d_{\max}(R) = \max_{l \in V} d_l(R)$, and the *minimum diameter* of R is $d_{\min}(R) = \min_{l \in V} d_l(R)$. When the region is understood from the context we drop the argument in the above notations and use, for example, b_l instead of $b_l(R)$. A

canonical trapezoidal region is of special interest, and is a canonical region that is quadrilateral and has exactly one pair of parallel sides.

The *canonical aspect ratio* $\mathsf{casp}(R)$ of canonical region R is the ratio of $d_{\max}(R)$ to $d_{\min}(R)$. In this paper, we use the terms aspect ratio and canonical aspect ratio synonymously

PBAR Trees, One-Cuts, and Two-Cuts. Given a *balancing factor* α, R is α-*balanced* or has a *balanced aspect ratio* if $\mathsf{casp}(R) \leq \alpha$. R is *critically balanced* if $\mathsf{casp}(R) = \alpha$. A cut c_l orthogonal to $l \in V$ that divides an α-balanced region R into R_1 and R_2 is *feasible* if both R_1 and R_2 are α-balanced.

Given a set S of n points in 2-dimensional space, a set of 3 partitioning vectors $V = \{\lambda, \mu, \nu\}$, a balancing factor α, $\alpha \geq 1$, and a reduction factor β, $0.5 \leq \beta < 1$, a *Parameterized Balanced Aspect Ratio* tree T is a BSP tree on S such that

1. All partitions are made with cuts orthogonal to the vectors in V;
2. The canonical aspect ratio of each region is at most α;
3. The number of points in each leaf cell of T is a constant with respect to n.
4. The depth of T is $O(\log_{1/\beta} n)$.

Given a balancing factor α and reduction factor β, an α-balanced region R is *one-cuttable* if there is a cut c, called a *one-cut*, orthogonal to a vector in V that divides R into two canonical subregions R_1 and R_2 such that

1. c is feasible;
2. $|R_1| \leq \beta|R|$ and $|R_2| \leq \beta|R|$.

(Note that if there is a continuum of feasible cuts that cover the entire region R, then R is one-cuttable, as at least one of these cuts will satisfy 2 above.) A region R is *two-cuttable* if there is a cut c, called a *two-cut*, orthogonal to a vector in V that divides R into two canonical subregions R_1 and R_2 such that

1. c is feasible;
2. $|R_1| \leq \beta|R|$;
3. $|R_2| \leq \beta|R|$ or R_2 is one-cuttable.

Shield Regions. Let R be an α-balanced canonical region and let x_l be a side of R, $x \in \{b, c\}$, $l \in V$. Now sweep a cut x'_l starting from the side opposite to x_l toward x_l. Let P be the subregion formed between x_l and x'_l. In the beginning, $\mathsf{casp}(P) \leq \alpha$. Sweep x'_l toward x_l and stop when P is critically balanced. P is called the *shield region* $\mathsf{shield}_{x_l}(R)$ of R with respect to x_l. x'_l is the *cut for* $\mathsf{shield}_{x_l}(R)$. R has two shield regions for each $l \in V$, $\mathsf{shield}_{b_l}(R)$ and $\mathsf{shield}_{c_l}(R)$. Note that R has a feasible l-cut if and only if $\mathsf{shield}_{b_l}(R) \cap \mathsf{shield}_{c_l}(R) = \emptyset$.

For a given $l \in V$, the *maximal shield region* $\mathsf{maxshield}_l(R)$ of R with respect to l is one among $\mathsf{shield}_{b_l}(R)$ and $\mathsf{shield}_{c_l}(R)$ that has the maximum size. (Remember, the size of a region is the number of points in it.) Note that R has a one-cut orthogonal to l only when $|\mathsf{maxshield}_l(R)| \leq \beta|R|$.

3 The PBAR Tree Algorithm

In this section we present the PBAR tree algorithm that, given a set of partitioning vectors $V = \{\lambda, \mu, \nu\}$, a reduction factor β, and a balancing factor α, constructs a PBAR tree on any set S of n points in 2-dimensional space; as long as $0.5 \leq \beta < 1$, and α is at least

$$f(V) = \frac{4.38}{\sin(\theta_{\min})\sin(\lambda\mu)\sin(\mu\nu)\sin(\nu\lambda)},$$

where $\theta_{\min}$ is the minimum among the angles $(\lambda\mu), (\mu\nu), (\nu\lambda)$.

The PBAR tree algorithm takes an initial α-balanced canonical region R that bounds S and recursively subdivides it by first searching for a one-cut, and if no such cut exists, by searching for a two-cut. For details see [8].

The algorithm for searching for a one-cut, OneCut, considers each partitioning vector in turn. For a partitioning vector l, a one-cut orthogonal to it exists if and only if the shield regions with respect to l don't overlap and the maximal shield region contains at most $\beta|R|$ points. Details are in [8].

The algorithm for searching for a two-cut, TwoCut, considers very few cuts as potential two-cuts. Only cuts for the maximal shield regions for the 3 partitioning vectors are considered as potential two-cuts. This is sufficient as long as $\alpha \geq f(V)$ — that this is true is our main result and we prove it in the rest of the section. Details for TwoCut are in [8].

Theorem 1 (Main Result). *Given a set S of n points in 2-dimensional space, a set of 3 partitioning vectors $V = \{\lambda, \mu, \nu\}$, a balancing factor α, $\alpha \geq f(V)$, and a reduction factor β, $0.5 \leq \beta < 1$, the PBAR tree algorithm constructs a PBAR tree on S in $O(n \log n)$ time.*

We first prove some preliminary lemmas. For all of these we shall assume that $0.5 \leq \beta < 1$ and $\alpha \geq f(V)$.

Lemma 1. *Given a set of partitioning vectors V, a balancing factor α, and a reduction factor β, if every α-balanced region R is two-cuttable, then a PBAR tree can be constructed for every set S of n points.*

Proof. Start with any initial α-balanced canonical region that bounds S. Since this region is two-cuttable, divide it into a maximum of 3 α-balanced subregions such that each contains less than βn points. Repeat this process for each of the resulting subregions until each of the final leaf regions has at most a constant number of points. The process, along any path of subregions, cannot be repeated more than $O(\log_{1/\beta} n)$ times.

Lemma 2. *A canonical region R that is a triangle is always α-balanced.*

Due to lack of space, proofs for the lemmas are in the full version of the paper[8].

Lemma 3. *Let $(l, m, n) \in \mathcal{P}(V)$. Let R be an α-balanced canonical region that is not critically balanced. If P is a critically balanced subregion created by partitioning R with an l-cut, then the minimum diameter of P is $d_l(P)$.*

Corollary 1. *For a critically balanced canonical region R that is a trapezoid, if b_l and c_l are the two parallel sides, then the minimum diameter of R is d_l.*

Lemma 4. *Let $(l, m, n) \in \mathcal{P}(V)$, and let R be an α-balanced region that has no feasible l-cut. Let S be the region formed by extending R such that $b_n(S)$ is of length 0. If P is $\mathsf{shield}_{c_m}(S)$, then $c_l(P) \leq c_l(R)$.*

Lemma 5. *Let $(l, m, n) \in \mathcal{P}(V)$. If an α-balanced region R has no feasible l-cut, then it has a feasible m-cut and a feasible n-cut.*

Corollary 2. *Let $(l, m, n) \in \mathcal{P}(V)$. If an α-balanced region R has no one-cut and no feasible l-cut, then the maximal shield regions with respect to m and n intersect.*

Lemma 6. *A critically balanced region R that is a trapezoidal is one-cuttable.*

Lemma 7. *Let $(l, m, n) \in \mathcal{P}(V)$. If an α-balanced region R has no one-cut and no feasible l-cut, then R has a two-cut.*

Lemma 8. *Let $(l, m, n) \in \mathcal{P}(V)$. If an α-balanced region R does not have a one-cut and yet there are feasible cuts along all 3 partitioning vectors l, m, and n, then R has a two-cut.*

Proof of Theorem 1. The PBAR tree algorithm recursively subdivides the initial bounding region by first searching for a one-cut, and if no such cut exists, by searching for a two-cut. By Lemma 1, if it always succeeds in finding a one-cut or a two-cut, it constructs a PBAR tree. It is easy to see that when the algorithm does not find a one-cut, no such cut exists. In such a situation, the algorithm searches for a two-cut by checking if any of the 3 maximal shield regions are one-cuttable. The proofs for Lemmas 7 and 8 show that at least one of these shield regions is one-cuttable, and so the algorithm always succeeds in finding either a one-cut or a two-cut. For the time analysis see proof in [8]. □

4 Empirical Tests

In this section we present the preliminary empirical results we have obtained and analyze them. We look at measures like number of nodes created, the depth of the tree, and the number of leaves visited instead of the actual time or space requirements. This is because the time and space measures are dependent on the efficiency of the implementation, the load on the machine during testing, etc. The other measures are not as dependent on the kind of testing carried out. In addition, the number of nodes visited is the dominant term if the data structure is stored in external memory (as is the case in SDSS).

We took 2 data sets and varied the partitioning planes in small increments and for each we found the best aspect ratio that can be obtained. First, we present plots that summarize the results of this experiment. Later, we present detailed results for 6 different data sets in which we compare BAR trees with $(0, \pi/4, \pi/2)$ partitioning angles with 2 instances of PBAR trees.

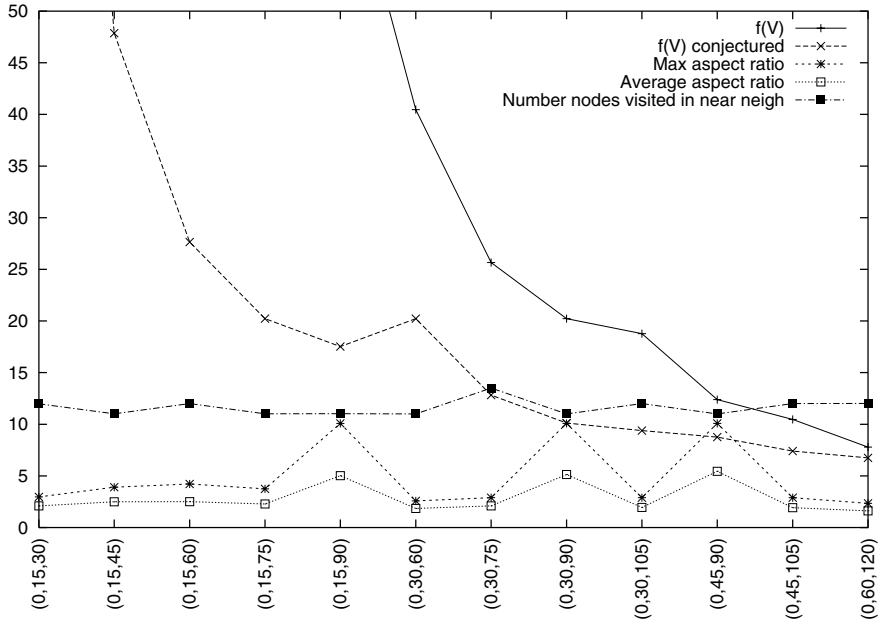

Fig. 2. Effects of varying the partitioning planes on SDSS data

4.1 Varying the Partitioning Planes

We varied the set of partitioning planes for a given dataset and found the best possible aspect ratio that can be obtained for that dataset. We plot this best empirical aspect ratio in Figures 2 and 3 (it is called the Maximum aspect ratio in the figures). We also plot, alongside, the bound $f(V)$ on the aspect ratio that we have proved. We had conjectured that $f(V)$ can be tightened by removing the $\sin(\theta_{\min})$ term to obtain $f(V) = 4.38/(\sin(\lambda\mu)\sin(\mu\nu)\sin(\nu\lambda))$. We plot the conjectured bound as well. We also plot the average aspect ratio of the nodes in tree.

The bound and the conjectured bound both decrease dramatically as the planes become evenly oriented. But the empirically obtained values do not follow their lead, though there is a reasonable amount of variation in the best (maximum) aspect ratio possible. The value of $f(V)$ converges towards the empirical value as it reduces, which indicates that it is possibly tight for the evenly oriented planes. The same is true for the conjectured bound; but, if you look closely at Figure 3 for the orientation (0, 45, 90) the conjectured value is actually lower than the best aspect ratio obtained through experiments. This indicates that the conjecture is wrong. We also plot the number of nodes visited during the nearest neighbor searches (the details for these searches are described in the next section) for the various planes. There is very slight variation in this, or in the average aspect ratio with the change in plane orientations.

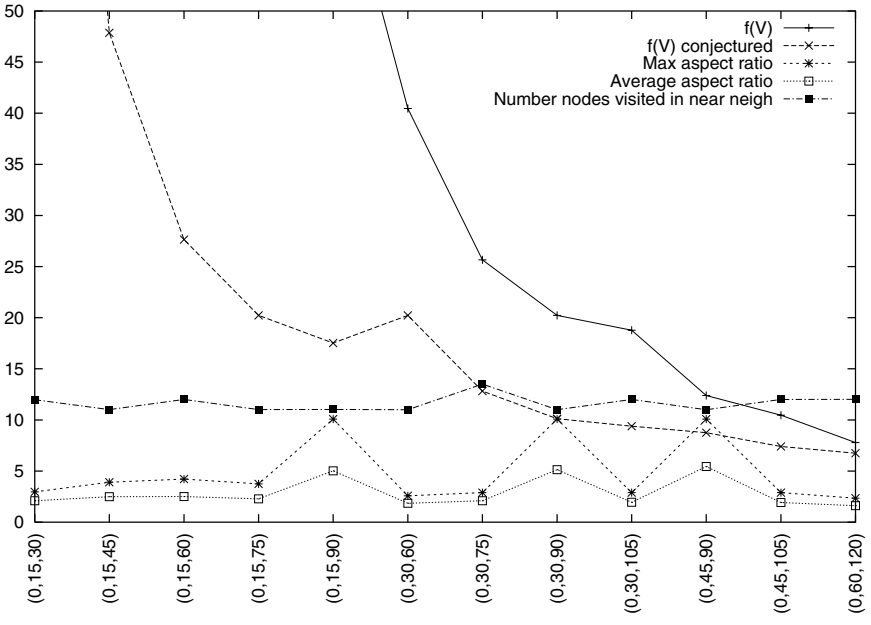

Fig. 3. Effects of varying the partitioning planes on data created uniformly at random along a circle

4.2 Comparing BAR Trees and PBAR Trees

To compare the performance of BAR trees (with mostly axis orthogonal planes: $0, \pi/4, \pi/2$) and PBAR trees we constructed PBAR trees with three different sets of partitioning vectors V. In the first set, V is such that $\theta_{\lambda\mu} = \pi/4$ and $\theta_{\lambda\nu} = \pi/2$. In the second set, V is such that $\theta_{\lambda\mu} = \pi/3$ and $\theta_{\lambda\nu} = 2\pi/3$, and in the third set, V is such that $\theta_{\lambda\mu} = \pi/6$ and $\theta_{\lambda\nu} = \pi/2$. Note that the partitioning vectors in the first set are that used by BAR trees. The PBAR trees constructed in this case closely mimic BAR trees and the performance is representative of the performance of BAR trees. In the second case, the partitioning vectors are more evenly oriented, while in the third case they are less evenly oriented, than the first case. We refer to the former case as the BAR tree results and the latter two cases as the PBAR-even tree and PBAR-uneven tree results respectively. In all cases α is 20, β is 0.6, and the maximum points in any leaf k is 5. A 100 $(1 + \varepsilon)$-nearest neighbor queries were solved using each tree. For each data set, a point q is first chosen uniformly at random from among the data points. This is the first query point. Then a random increment is chosen and repeatedly added to q to get 99 other query points. ε for the queries is always 0.001. The results for BAR tree are in Figure 4, for PBAR-even tree are in Figure 5, and for PBAR-uneven tree are in Figure 6. Of the 6 data sets data sets, 4 were chosen rather arbitrarily, and the last two are real data from the Sloan Digital Sky Survey(SDSS). The data sets are described in [8].

Data Set	Nodes in tree	Depth of tree	Avg. $\mathsf{casp}(\cdot)$ of regions	Nodes visited during query	Leaves visited during query
Set 1	4615	12	9.70	57.2	11.74
Set 2	4719	13	8.78	73.64	13.97
Set 3	4709	12	9.08	148.51	30.97
Set 4	4631	12	7.84	16.47	2.49
Set 5	4749	12	8.98	13.49	1.60
Set 6	2079	11	9.13	21.84	3.68

Fig. 4. Results for $(1+\varepsilon)$-nearest neighbor queries on BAR trees. (Number of nodes and leaves visited during query are averaged over a 100 queries.)

Data Set	Nodes in tree	Depth of tree	Avg. $\mathsf{casp}(\cdot)$ of regions	Nodes visited during query	Leaves visited during query
Set 1	4647	12	9.93	21.5	3.66
Set 2	4791	12	8.78	22.61	4.38
Set 3	4741	12	9.36	13.7	1.48
Set 4	4641	12	8.14	15.59	2.23
Set 5	4673	12	8.41	32.44	5.91
Set 6	2067	11	8.77	21.19	3.49

Fig. 5. Results for $(1+\varepsilon)$-nearest neighbor queries on PBAR-even trees; V such that $\theta_{\lambda\mu} = \pi/3$ and $\theta_{\lambda\nu} = 2\pi/3$. (Number of nodes and leaves visited during query are averaged over a 100 queries.)

Data Set	Nodes in tree	Depth of tree	Avg. $\mathsf{casp}(\cdot)$ of regions	Nodes visited during query	Leaves visited during query
Set 1	4635	12	9.64	58.04	12.03
Set 2	4697	13	8.58	76.32	14.91
Set 3	4655	12	8.86	152.81	31.62
Set 4	4629	12	7.44	15.96	2.37
Set 5	4683	12	8.57	13.48	1.6
Set 6	2077	11	8.82	21.21	3.51

Fig. 6. Results for $(1+\varepsilon)$-nearest neighbor queries on PBAR-uneven trees; V such that $\theta_{\lambda\mu} = \pi/6$ and $\theta_{\lambda\nu} = \pi/2$. (Number of nodes and leaves visited during query are averaged over a 100 queries.)

The number of nodes in the tree are about the same, for the first 5 data sets, irrespective of V. This is expected as the number of data points and the maximum size of a leaf are the same in all cases. The depth of the trees are about the same too, irrespective of the data set. This, again, is expected as the β values are the same, and we don't except too many regions that require two-cuts. Set 6 has far fewer points and so has much fewer nodes. Surprisingly, the

average values of the canonical aspect ratio are not very different in the three trees for the different data sets. Neither is one of the trees always better than the other. Such is not the case for number of nodes and number of leaves visited during query processing. PBAR-even trees almost always visit fewer nodes and fewer leaves and in one particular case the difference with both BAR and PBAR-uneven trees is a factor of 10. That PBAR-even trees perform better at approximate nearest neighbor searches may be expected given the theoretical results, it is surprising that this should be the case when the canonical aspect ratio values are about the same. For Set 5, which is real data set from the SDSS, however, PBAR-even trees are not the best. For this set, both BAR trees and PBAR-uneven trees perform better, with PBAR-uneven slightly ahead of BAR. For the other real data, Set 6, again PBAR-even is the best.

In conclusion, our experiments show that the flexibility of PBAR trees can help in increasing the efficiency of approximate nearest neighbor searches.

5 Conclusion and Future Work

In this paper we revisited BAR trees in 2 dimensional spaces and developed the Parameterized Balanced Aspect Ratio (PBAR) trees. These allow any given set of 3 partitioning planes and yet retain all the advantages of BAR trees — general purpose data structures, space efficient, logarithmic depth, bounded aspect ratio, and poly-logarithmic approximate query processing. These are the first known "BAR-type" trees in which the aspect ratio can be bounded by a simple closed-form function (it depends on the orientation of the partitioning planes). We conducted empirical tests that show that in many instances the evenly oriented partitioning planes are better than the mostly axis orthogonal planes that have been mostly studied prior to this. In addition, our experiments indicate that our bound is tight in some respects: it converges to empirical values for evenly oriented planes, and that a natural modification to tighten it (our conjecture that $\sin(\theta_{\min})$ factor can be removed) is wrong.

Having bounds on the aspect ratio can be useful in ways other than solving queries faster. For example, PBAR trees can be used to efficiently compute the density of a region around a given point. This can be useful in detecting density based outliers. We want to explore this and other possible applications of PBAR trees in spatial data mining.

Acknowledgment

The authors will like to thank Breno de Medeiros for many helpful suggestions, including helping tighten the bounds in Lemma 7 and Christian Duncan for helpful discussions.

References

1. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.
2. Sunil Arya and David M. Mount. Approximate range searching. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 172–181, 1995.
3. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.
4. J. L. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Eng.*, SE-5:333–340, 1979.
5. J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 187–197, 1990.
6. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.
7. Paul B. Callahan and S. Rao Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 263–272, 1995.
8. A. Chaudhary and M.T. Goodrich. Balanced aspect ratio trees revisited. http://www.cse.nd.edu/˜achaudha/research. Full version of the paper.
9. M. Dickerson, C. A. Duncan, and M. T. Goodrich. K-D trees are better when cut on the longest side. In *Proc. 8th European Symp. on Algorithms*, volume 1879 of *Lecture Notes Comput. Sci.*, pages 179–190. Springer-Verlag, 2000.
10. C. Duncan. *Balanced Aspect Ratio Trees*. PhD thesis, The Johns Hopkins University, Baltimore, Maryland, Sep 1999.
11. C. A. Duncan, M. T. Goodrich, and S. Kobourov. Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. In *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Alg.*, pages 300–309, 1999.
12. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. In *Graph Drawing*, Lecture Notes in Computer Science, pages 111–124. Springer-Verlag, 1998.
13. H. Fuchs, G. D. Abrams, and E. D. Grant. Near real-time shaded display of rigid objects. *Comput. Graph.*, 17(3):65–72, 1983. Proc. SIGGRAPH '83.
14. H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.
15. David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.
16. Robert Lupton, F. Miller Maley, and Neal Young. Sloan digital sky survey. http://www.sdss.org/sdss.html.
17. Robert Lupton, F. Miller Maley, and Neal Young. Data collection for the Sloan Digital Sky Survey—A network-flow heuristic. *Journal of Algorithms*, 27(2):339–356, 1998.
18. H. Samet. *Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods*. Addison-Wesley, Reading, MA, 1989.
19. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.