Intuition for the algorithm:

Locate the leaf node containing the query point (log n time)

Enumerate leaf nodes in increasing order of distance.

When a cell is visited, the distance from q to the point in the cell is computed. We keep track of the closest point seen so far (p). Now we keep traversing other cells in the priority list.

If the distance to the current leaf cell exceeds dist (q, p) / (1 + e), it follows that the search can be terminated and p can be reported as an approximate near neighbor of q.

The number of cells visited is a function of d and e (and not n and r).

So the time is given as O(log n + f(d)).

The algorithm can be extended to k near neighbors by successively removing the closest approximate point and repeating the search. This does it in time O(k log n).

Balanced Box Decomposition (BBD Trees)

O(log n) height

Subdivides space by axis alighed rectangles (the ratio of the shortest to longest is boounded).

Space is recursively subdivided into collection of cells.

Each cell is a d-dimensional rectangle

Each node in the tree is associated with a cell
(and hence implicitly with a set of points in the cell)

Each leaf cell is associated with a single point lying within the bounding rectangle of the cell.

The tree can be built in dn log n time and has O(n) storage.

Approximate Near Neighbor Searching in Fixed Dimensions

(S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, ``An optimal algorithm for approximate nearest neighbor searching," 5th Ann. ACM-SIAM Symposium on Discrete Algorithms, 1994. *http://www.cs.umd.edu/%7Emount/pubs.html*)

Given n data points in a real d-dimensional space and a query point q, the near-neighbor of q is the data point that is closest to q.

An e-accurate near neighbor is within (1 + e) of the near neighbor.

With n log n preprocessing and O(n) storage, we can compute the e-accurate near neighbor in time log n and k near neighbors in time k log n.

In 1 dimension, we can sort and do a binary search.

In 2 dimensions, we can compute the voronoi diagram for the point set and use fast planar point location algorithms to locate the cell containing the query point.

For d > 2, voronoi diagrams scale as O(n^(d/2)).

For regular grids in higher dimensions, we can achieve good average case performance.

(grid based methods).

Query Algorithms

Similarity Selection or Similarity Search Algorithm

Traverse tree eliminating intermediate nodes outside the sphere

If leaf in sphere, insert into list in ordered fashion.

Improved techniques:

The algorithm searches subtrees in increasing order of minimum distance from the query, and stops searching when the closest subtree is outside of the current search region (region that must be searched to satisfy the query). If a subtree is a non-leaf node, then all children that might need to be searched in the future are added to the search priority queue. If the subtree is a leaf node, both the result set and the search region are updated, if needed, using the objects in the leaf.

k-d tree Variants

Sproull suggested splits on the principle axis or eigenvector (rather than on one dimension) for highly correlated datasets. The problem with this approach is that splits are not orthogonal.

Other variants of k-d tree splitting algorithms use (1) split orientation based on the variance, and the (2) split position approximately the median.

Optimized Structures for Medium and High Dimensional Datasets

Three dimensions of k-d tree variations:
(1) split (partition plane) orientation,
(2) split (partition plane) position, and
(3) distance representation.

Although other variants and refinements have been suggested sincethen, most still fall into this general framework.

The original optimized k-d tree used the dimension with the greatest spread as the split dimension (1), chose the split position to be the median (2), and calculated distance using bounds on all dimensions(3).

Other tree structures augment the values with min and max values.

Some tree structures collapse the box to the min and max values.

Dynamic vs. Non-Dynamic Updates

Non-dynamic structures tend to be faster. On the other hand, dynamic structures offer significantly higher flexibility.

Limitation of dynamic structures based on the k-d tree

The problem is simply that structures based on the k-d tree are dependent on the distribution of the vectors that are initially inserted into the structure. The smaller initial dataset is used to determine the splits that are later used to index the complete dataset. The split locations can not be changed, since doing so could require the reorganization of all elements in the subtree. In fact, for high dimensional datasets, changing a split would almost certainly require the reorganization of all buckets in the subtree. If the initial dataset does not have a distribution that is representative of the distribution of the final dataset, then the chosen splits will almost certainly be suboptimal.

This effect is reduced by allowing the original k-d tree structure to become unbalanced, so that the effects of bad initial split choices can be reduced by allowing splits to propagate higher in the tree.

Approximation: Is it really useful?

Improved performance with negligible error.

Increased Flexibility in Query Specification.

Increased Flexibility in System Design.

Incremental Approximate Query Processing

The idea is that the database server (or query processing module) will incrementally send the client (or user interface module) query results that are incrementally better approximations of the exact query result. This could free the user of having to specify a priori the level of approximation desired. If the user does not issue any new queries, the approximate results will be displayed first, and later the exact query results will eventually be displayed. If the user issued a new query based on the approximate result before the query is complete, then query processing would be terminated without ever calculating the exact result.

Approximation

Measures of Approximation

Worst Case Error: The first measure of approximation is in terms of the worst case approximation allowed. Intuitively, e is the worst case error in the distance, measured in terms of the exact result, that the search algorithm is allowed to make.

More formally: $D/D' <= 1 + e$

Here D is a point picked which shouldnt have been and D' is the one that should have been.

Probability of Error: This measure of approximation error is much more intuitively meaningful than the worst case error described above. It simply measures the probability that the approximate result is different from the exact result, over the expected query distribution.

Average Error: This measure of approximation error is also more meaningful from the user's perspective. It measures the average rather than the worst case error.

Some observations:

1.As a rule of thumb, search time increases exponentially with the intrinsic dimension of the dataset. On uniform distributions, the exponential dependence on dimension occurs for all known (practical) algorithms on a uniformly distributed dataset. However, it is difficult to measure the intrinsic dimension of a real dataset, and perhaps the best way to measure this is to compare the performance of a dataset with the performance of uniform distributions.

2.Eliminating or minimizing overlap between regions in a spatial data structure is critical to provide good performance on medium or high dimensional datasets in most applications.

3. Support for approximate k-nearest neighbor and approximate (spherical) range searching algorithms should be provided in a general purpose indexing structure for medium or high dimensional datasets

Nearest neighbor searching:

The k-d tree is probably the data structure most often used in practice for nearest neighbor searching in main memory. It has been shown that the optimized k-d tree requires logarithmic expected time with respect to the size of the dataset (ignoring dimension).

Recent work in computational geometry on nearest neighbor searching has focused on the approximate nearest neighbor and k-nearest neighbor problem. It has been observed that for medium or high dimensional datasets, even allowing a small probability of an incorrect result can provide a noticeable speedup, and when some error can be tolerated, larger speedups (sometimes an order of magnitude improvement) can be achieved.

Researchers have provided optimal worst-case algorithms for approximate nearest neighbor searching (with respect to the size of the dataset). For the approximate nearest neighbor problem, a worst case optimal algorithm (with respect to the dataset size denoted as n) requires O(n log n) time to pre-process, takes O(n) storage and can satisfy queries in time O(log n).

Nearest neighbor searching:

Perhaps the first algorithm proposed for nearest neighbor searching is the bucketing algorithm or Elias's algorithm. This simple algorithm uses an regular partitioning of the space containing the dataset (for instance, in a grid), and each partition is assigned a bucket which holds all points lying within it. Searching for nearest neighbors is accomplished by searching the buckets in order of their closest distance from the query point until all buckets have been searched that could contain a closer nearest neighbor.

This algorithm provides good performance on uniformly distributed data, but is not practical for most non-uniform distributions used in practice. This algorithm was analyzed by Rivest and later by Cleary. Cleary's analysis shows that the algorithm on average requires time exponential O(2^d) with dimension, but is independent of the size of the dataset, assuming a uniform distribution and a sufficiently large dataset (N >> 2^d).

The similarity selection operation:

The similarity selection operation is a generalization of the k-nearest neighbor query and spherical range query, and approximate versions of those queries. The parameters of the similarity selection or similarity search query operation are the following:

1. A query object Q . The query results are ordered in increasing distance from Q, where distance is defined by some distance function D(x,y) where x and y are objects.

2. A positive integer k that specifies the maximum number of (approximate) nearest neighbors returned in the query result.

3. A positive real number T that specifies the maximum distance from the query vector of any vector in the query results. Note that if T is larger than the distance from to the kth nearest neighbor (ie. T is large), then an exact query returns the k-nearest neighbors of Q. If k is larger than the number of objects within distance T of (ie. k is large), then an exact query becomes a spherical range query bounded by T relative to Q.

4. A parameter that specifies the amount of the approximation error allowed.

Similarity-Based Queries

Similarity Selection: “Find objects similar to a reference.” This query finds exact or approximate k-nearest neighbors or all near neighbors within a threshold distance (range query). We also call it a similarity search operation.

Similarity Join: “Find all similar sets of objects.” This query is often called an all-pairs query because it usually finds all pairs of objects that satisfy the constraints of a similarity selection, where the pair either comes from the same set or two sets of objects, although in general this operation could find all-triples, etc. This type of join operation has been used recently in time series database applications.

Similarity Sample: “Sample objects similar to a reference.” This query returns some type of sampling of the dataset relative to a reference. This might be useful as a browsing operation to “zoom in” and ``zoom out” of the database.

Assume that for any two objects x and y some type of distance function D(x,y) is defined between them that measures the dissimilarity between objects.

More General Models: In some applications, neither of the two models described above will be appropriate. A number of researchers have pointed out that perceptual, cognitive and ``intuitive" notions of similarity do not adhere to the metric space model of similarity, much less the vector space model. However, the problem is that some assumptions are required in order to allow objects to be indexed. If sublinear access via an index is desired, the practical solution will often be to use a distance function that satisfies the metric assumptions and approximates the true distances between objects for indexing.

Weighting plays an important role.

Metric Space Model: The vector space model is too restrictive in some applications. In these applications, it is difficult or impossible to map each object into a point in d-dimensional vector space while still accurately representing the similarity or distances between objects using some metric in that space.

By definition, a metric distance function D has three properties:

1. Positivity: $D(x,y) >= 0$ and $D(x,x)=0$
2. Symmetry: $D(x,y) = D(y,x)$
3. Triangle Inequality: $D(x,z) <= D(x,y) + D(y,z)$

This is much more restrictive than the Vector Space Model, since no assumptions can be made about the representation used for calculating similarity. Therefore, useful operations such as splitting on a dimension, bounding a set of objects using a hyperrectangle, or calculating the centroid of objects are not possible, because the object representation is opaque. Of course, any indexing method designed for the Metric Space Model can be used to index objects in the Vector Space Model.

Similarity Models for Indexing

Based on some time of type of indexing so that only a subset of the objects need even be considered, resulting in a sublinear similarity search algorithm.

Model name	Object Rep.	Index Str.	Pros/Cons
Vector Space Model	Fixed size vector	k-d tree, R-tree	fast / inflexible
Metric Space Model	Variable Length (symmetric, positive, triangular distance)	B-K tree / GNAT	slower / flexible
More General Models	Variable Length	Non-standardized	slower still

Vector Space Model: In this model, the assumption is that each object can be fully or partially represented as a point in a d-dimensional vector space, where d is fixed. The dissimilarity between two objects is measured using some metric distance function over d-dimensional space. The most common metric distance function is Euclidean distance, but other metrics such as Manhattan, or Max can be used.

This is the most common model used for indexing multimedia databases. (aka F-index approach).

Algorithms and Strategies for Similarity Retrieval
(http://vision.ucsd.edu/papers/simret/journ1.html)

Retrieval of objects based on similarity.
Based on the Vector Space Model where each object is represented as a vector of fixed dimension

Applications:

Information Retrieval: In the vector space model of information retrieval, similarity retrieval extends into the domain of documents where the “object” is a document, and queries attempt to find documents that are “semantically” similar to a user’s query. In more recent work, latent semantic indexing (LSI) has been suggested , which represents each document by a vector with 100-400 dimensions. This representation usually provides better retrieval performance than the standard vector space model by taking advantage of associations between terms.

Time Series and Sequence Databases:

Case-based Reasoning: In case based reasoning, an AI system often uses previously recorded similar cases to reason about a new problem. For example, automatic real estate appraisal programs [Gonzalez and Laureano-Ortiz, 1992] must find comparable or similar real estate recently sold in order to estimate the current value of the real estate of interest.

Clustering: Many clustering algorithms (k-means clustering and hierarchical clustering) rely on efficient algorithms to compute similar objects or pairs of similar objects (closest point problems).