CoverTrees

The main reason why cover trees were introduced in [Langford et al., see http://hunch.net/~jl/projects/cover_tree/cover_tree.html] was as a data structure for quickly searching nearest neighbors of a point, or points within a certain distance of a point, when data is intrinsically low-dimensional in a suitable sense.
Finding the nearest neighbors is a basic task that is needed in a wide variety of machine learning algorithms, including many classification, regression, clustering and manifold learning algorithms.
The naive algorithm for computing k nearest neighbors to a query point among n given points requires n distance calculations: compute the distance between the query point and each given point, sort, return the indices of the points corresponding to the k smallest distances. This is OK for one query point, but not for many query points, for example if we have n query points, this scales like n^2. This is not an uncommon situation, for example in manifold learning one often needs to connect every one of n points to its k-nearest neighbors, therefore requiring, with the naive algorithm, n^2 distance calculations.
Cover trees have several nice property that they can be constructed in order n\log n on n points, and yet they allow one to find the k nearest neighbors of a query point in order \log n. The basic idea is that cover trees have the property that the set of points C_j at scale up to j in the tree (i.e. at distance, on the tree, up to j from the root of the tree) form a well-separated net of points: no two points in C_j are closer than 2^(-j) and for each point in the space there is a point in C_j within distance 2*2^(-j) [here we assume that the space has diameter 1]. Then in order to find nearest neighbors we can use a branch-and-bound strategy where the query point is fed at the root of the tree, and then if the distance of that point to a child of that node is too large (roughly 3/2*2^(-j)), then the point will not go down the paths that include the descendants of that child, but only down the other paths. In this way the number of paths the points has to descend stays of order 1 at all scales, and since there are log(n) scales, this operation will require only log(n) operations. Warning: details matter in all of the above, this is a very crude description. An important remark: the constants in front of the costs above (n\log n for the construction of the cover tree, and log n for the nearest neighbor search) are exponential in the intrinsic dimension of the space. So if the data is on a d-dimensional manifold in R^D, with d much smaller than D, the cost depends exponentially in d but not on D. Moreover, the definition of intrinsic dimension is quite robust. The knowledge of d is not required by the algorithm, which is in this sense adaptive. Finally, such dimension is in principle allowed to change from scale to scale and from location to location, which is useful in the case of high-dimensional data sets. For example even if the data is exactly on a d-dimensional manifold, but is contaminated by D-dimensional noise, than at all scales larger than the scale of the noise the cover tree will be behave as if the data is d-dimensional, even if technically its distribution is D-dimensional.
Another nice property of the cover tree algorithm is that it is incremental: the tree is constructed by adding points (e.g. one at a time or in batches) and each addition only grows the tree but does not require pruning or rewiring operations.