Summary: BIRCH: An Efficient Data Clustering Method for Very Large Databases (1996) Tian Zhang, Ragu Ramakrishnan, Miron Livny (all at U. of Wisconsin-Madison at time of publication)

**Main point:** The authors describe a hierarchical clustering method, which uses a new data structure similar to a B-tree, called a CF-tree, to store a small amount of information about each cluster in order to dynamically update clusters in a linear scan of the dataset. The algorithm is further optimized by removing outliers efficiently. BIRCH assumes that points lie in a metric space and that clusters are spherical in shape.

The CF-tree is composed of CF nodes, where CF stands for “clustering feature.” A clustering feature $CF_i$ is simply a triple \{N_i, LS_i, SS_i\} where $N_i$ is the number of points in the cluster represented by $CF_i$, $LS_i$ is the linear sum: $LS_i = \sum_{i=1}^{N} x_i$, where $x_i$ is a point (vector) that has been assigned to cluster represented by $CF_i$, and $SS_i$ is the square sum of the data points $x_i$ in the cluster represented by $CF_i$: $SS_i = \sum_{i=1}^{N} x_i^2$. Because each of the components of $CF_i$ is a linear sum, we have the additivity property for each $CF$: $CF_i + CF_j = \{N_i + N_j, LS_i + LS_j, SS_i + SS_j\}$. Importantly, we can also calculate intra- and inter-cluster distances using the CF nodes. For example, what the authors call “centroid Euclidean distance” $D_{0ij}$ is defined as the Euclidean distance between the centroids of two clusters $i$ and $j$:

$$D_{0ij} = \left( \left( \frac{\sum_{m=1}^{N_i} x_m}{N_i} - \frac{\sum_{l=1}^{N_j} x_l}{N_j} \right)^2 \right)^{\frac{1}{2}}$$

Note that this is the same as:

$$D_{0ij} = \left( \left( \frac{LS_i}{N_i} - \frac{LS_j}{N_j} \right)^2 \right)^{\frac{1}{2}}$$

The radius, $R_i$, diameter $D$, defined as:

$$R_i = \left( \frac{\sum_{m=1}^{N_i} \left( x_m - \frac{\sum_{l=1}^{N_i} x_l}{N_i} \right)^2}{N_i} \right)^{\frac{1}{2}}$$

$$D_i = \left( \frac{\sum_{m=1}^{N_i} \sum_{l=1}^{N_i} (x_m - x_l)^2}{N_i(N_i - 1)} \right)^{\frac{1}{2}}$$

and other inter-cluster distances can be similarly expressed the terms $N_i, LS_i$, and $SS_i$. 

1
Figure 1: An example of a CF-tree, which stores three pieces of information per cluster: its size, a linear sum of its elements and a sum of its elements squared. The leaf nodes represent the finest granularity of clusters, and each non-leaf node represents a cluster made up of the subclusters represented by its entries.

The CF-tree is height-balanced with a branching factor $B$ and a threshold distance $T$. Each nonleaf node contains at most $B$ nonleaf entries of the form $[CF_i, \text{child}_i]$, where $\text{child}_i$ points to the $i$-th child node. A nonleaf node represents a cluster made up of all the subclusters represented by its entries, similarly to a single linkage dendrogram. A leaf node contains entries of the form $[CF_i]$, where $i = 1 \ldots L$, and the leaf nodes are linked together by forward and back-pointing pointers. All entries in a leaf node must satisfy the property that the diameter (or radius, depending on set parameters) of the cluster represented by the entry $CF_i$, must be less than $T$. A schematic of the CF-tree is shown in Figure 1.

The CF-tree is built dynamically, following similar insertion rules as a B-tree. A new cluster is inserted into the tree recursively, finding its closest child node by computing a distance function such as $D_{ij}$ above using the clustering features $CF_i$. If adding the cluster to its closest leaf node causes the number of entries in the leaf to exceed $L$, then the leaf node is split by choosing the farthest two clusters by the same distance metric to be the new seeds, and redistributing the entries according to their closeness to the two new seeds. Splitting a leaf node may cause splits of nonleaf nodes all the way up the tree, because with each split we need to store a new $CF$ entry representing the new node. A merging step is added after each split, where the nonleaf node which has just added a new $CF$ entry to accommodate the split is scanned to see if any two entries within the node can be merged into one cluster.

After the CF-tree is built, an optional Phase 2 attempts to build a smaller CF-tree by removing outliers and grouping subclusters into larger clusters, and
a “global clustering” is applied to the leaf nodes in Phase 3. The authors note that many well-known algorithms can be applied in this phase to work on the CF-tree leaf nodes. They chose to use single linkage clustering. Finally, the optional Phase 4 attempts to refine the clustering by employing a $k$-means like algorithm, choosing the centroids of the clusters produced by Phase 3 as seeds, and moving nodes to the cluster whose centroid is closest to them.

Because the choice of threshold value $T$ will determine the size of the tree, an attempt is made to increase $T$ and rebuild the tree dynamically if the tree gets to be too large to fit into memory. The authors suggest to set the threshold “conservatively”. When re-building the CF-tree using a higher threshold, choosing how much higher to set the new threshold is a difficult problem in general, and the authors solve it by using heuristics.

The time complexity of Phase 1, the CF-tree building phase, is:

$$O\left(dNB\left(1 + \log_B\left(\frac{M}{F}\right)\right) + d\frac{M}{ES} \log_2\left(\frac{N}{N_0}\right)B\left(1 + \log_B\left(\frac{M}{P}\right)\right)\right)$$

where $B$ is the branching factor, $ES$ is a $CF$ entry size, $\frac{M}{P}$ is memory divided by the page size, which corresponds to the maximal size of the tree, $d$ is the dimension of data points, $N$ is the total number of data points, and $N_0$ is the number of data points loaded into memory with threshold $T_0$.

Experimental results are shown for simulated data, where clusters are of different sizes and configurations in two dimensions. BIRCH is shown to be about 15 times faster than CLARANS, the state-of-the-art in large-scale clustering at the time. BIRCH also finds clusters accurately – the authors show that the number of points in a BIRCH cluster is no more than 4% different from the corresponding true cluster. Parameter settings are also tested and reported for the choice of threshold $T$, whether the optional outlier pruning phase is enabled, etc. The choice of $T = 0$ works well if the user has no prior knowledge of the data, since the $CF$-tree will automatically select a higher threshold if it runs out of memory. The outlier handling phase seems to work well, improving the accuracy as well as runtime of BIRCH. The choice of maximal tree size shows that BIRCH can trade off memory and time to achieve similar quality.

BIRCH is also tested with real data which comprises two images, and performs fairly well. The authors call its performance on real data “satisfactory.”

The authors conclude with future work directions: 1) better ways to dynamically adjust the threshold, 2) dynamic adjustment of outlier criteria, 3) more accurate quality measurements, and 4) data parameters that are good indicators of how well BIRCH is likely to perform. I think that (4) is interesting because BIRCH assumes a spherical cluster shape, so it would be interesting to test/adjust it on/for other cluster topologies. (CURE claims to do this well)