Fast Dimensionality Reduction for Brain Tractography Clustering

January 29, 2010

1 Introduction

Algorithms for clustering and classifying diffusion imaging white matter tractographies are typically quite slow often needing days or weeks of processing on a single CPU core Much of the computational burden of clustering techniques arises from the need for detailed geometric comparisons between pairs of tracks in large datasets often containing hundreds of thousands of tracks. We have developed an approach that leaves this more detailed comparison to a later stage after an initial first pass through the dataset to create prototype bundles. We present a fast method, which in less than 5 minutes generates preliminary clusters from a whole brain tractography dataset of 250,000 tracks. Our algorithm is inspired by the BIRCH algorithm (Zhang et al. xxxx). When clusters are held in a tree structure this permits upwards amalgamations to form bundles out of clusters, and downwards disaggregation to split clusters into finer sub-clusters corresponding to a lower distance threshold.

2 Methods

Current high definition fiber tracking methods can produce about 300,000 tracks. A track is a curve simulating neural fibers consisting of up to several hundreds of line segments. To reduce the number of searches in this massive dataset we generate from dataset a graph where the nodes consists of a virtual (representative) low dimensional track, the number of tracks in the cluster and the indices of the tracks in the cluster. This virtual track is the mean of all the downsampled tracks in the node. For the downsampling we found we could get useful results by approximating a track with just 2 directly connected line segments.

This first pass method is based on the observation that for two tracks to be considered similar we expect at least a corresponding start, end, and middle points of the tracks to be close to each other. Each track in the dataset is approximated by a three-point track (two ends and the middle). If SP, MP and EP are the start, middle and endpoints, then 3TED = min(|SP1-SP2|+|MP1-MP2|+|EP1-EP2|, |SP1-EP2|+|MP1-MP2|+|EP1-SP2|)/3. We generate clusters of 3-tracks using a fast agglomerative hierarchical clustering algorithm using the 3TED distance metric. As we create clusters, we generate the virtual track (r) for the cluster, given by the centroid of the constituent 3-tracks. The algorithm consists of two phases. In the split phase:

1) Select the first track t_1, and place it in the first cluster r_1={t_1}

For all remaining tracks n where 2<n<=N (where N is the number of tracks):

2) Select next track t_n.

3) Calculate 3TED between this track and virtual tracks of all current cluster (r_m where 1<=m<=M and M is the current number of clusters).

4) Add the track to the cluster with the minimum 3TED, and update r_m=r_m union {t_n} as long as the minimum distance is smaller than a specified threshold, otherwise create a new cluster with 1 track.

In the merge phase we create a higher node that aggregates nearby clusters by comparing their virtual 3-tracks. The new cluster is the union of the two previous clusters.

3 Results

Figure 1 shows how the algorithm performs in clustering a bundle from the Fall 2009 Pittsburgh brain competition (PBC) (http://pbc.lrdc.pitt.edu). The bundle consisted of the 1076 tracks labeled by the neuroanatomist as being in the fornix. The first panel shows all the tracks in white. The rest of the figure shows detected clusters, with tracks in a cluster sharing the same unique color. The top right panel shows the results of our algorithm with a distance threshold of 5mm. There are 22 clusters. Left and right clusters are distinct. There are different clusters for short and long groups of tracks. The bottom left panel shows the 7 clusters found with a distance threshold of 10mm. The left and right long bundles remain distinct, but the central part of the fornix now has a single main cluster. The bottom right panel shows the single cluster that found with a distance threshold of 20mm.

Figure 2 shows the result of our method for the whole brain from the first PBC track dataset, consisting of 250K tracks. The left panel shows all the tracks in white. The middle panel shows the 158 clusters that result from whole brain clustering with a distance threshold of 20mm. There was plausible differentiation between bundles - for example note the well-differentiated descending corticospinal tracks. On the right we show the corresponding virtual 3-tracks. It took around 5 minutes to run whole-brain clustering on one core of a 2.5 GHz Intel PC.

The algorithm is online in that it does not require that all the data is available at the outset but additional tracks can be incorporated later with the clusters being automatically updated. This is useful for creating an average track dataset combining datasets for several brains. It also supports a multiresolution representation of the tractography. The graph structure for holding the cluster information can be either a tree or a graph whose nodes have links to their closest neighbours. In cases where the distance threshold is very low we can use this latter graph representation to increase our search speed.

4 Conclusions

Our method reduces the search space between tracks in large trajectory datasets from tractography. The algorithm has has very low computation time and memory use. It may be used for making a first pass clustering, to reduce the number of detailed comparisons between full track descriptions. Our method is hierarchical; clusters can be split into sub-clusters by decreasing the distance threshold. Making a graph of the cluster structure can be rapidly traversed to look for similarity of clusters across different scales.

The results here use only three points (the start, middle and end point). This is not intrinsic to our technique; we can use more points to approximate the tracks, and different distance measures (Zhang 2xxx; Jianu, 2009), to detect similarity. More detailed approximations consisting of more segments can be used at a later stage.

The Python / Cython code for our development work is published in the open-source dipy project, hosted at http://github.com/matthew-brett/dipy.