|  |
| --- |
| BIOINFORMATICS  Population-scale Three-dimensional Reconstruction and Quantitative Profiling of Microglia Arbors  Murad Megjhani1,\*, Nicolas Rey1, Amine Merouane1, Yanbin Lu1, Amit Mukherjee3, Kristen Trett2, Peter Chong2, Carolyn Harris2, William Shain2, Badrinath Roysam1  1Department of Electrical and Computer Engineering, University of Houston, Houston, TX.  2Center for Integrative Brain Research, Seattle Childrens Hospital, Seattle, WA 98101.  3Cold Spring Harbor Laboratory, Cold Spring Harbor, NY 11724  Received on XXXXX; revised on XXXXX; accepted on XXXXX  Associate Editor: XXXXXXX |

# Supplementary Material

## Calculation of Q and H for Sparsity Based Modeling of Microglia Processes

(3)

The second termrepresents the discriminative sparse-coding error. It enforces that the sparse codes approximates the discriminative sparse codes. Intuitively it forces the signals from the same class to have similar representation. are the discriminative sparse codes of input signalsfor classification that are calculated based on the number of labels. For example, ifis the discriminative sparse code corresponding to the input signal, then the non-zero values of occur at those indices where the input signal and the dictionary element share the same label. For example, for the given problem (2 class problem), assumes that the size of the dictionary is 8 i.e. and the number of examples is 10 i.e.. To create a discriminative dictionary we want that the examples of the class 1 () to use the first four dictionary elements i.e. and the rest of the elements to be used for class 2. Then can be defined as

The matrix is set automatically and user doesn’t have to set this explicitly.

The third term represents the classification error, are the class labels for the input for classes. For the above example of ten inputs with 2 classes, can be defined as

## Priority-queue Based Reconstruction Algorithm

In order to implement our modified version of Prim’s algorithm, we use a priority queue based approach. Each element of the queue is a voxel in the original input image. Both primary and secondary nodes are added to the queue. The priority of each element is determined by its cost, with lower costs indicating a higher priority. The queue is initialized with the root nodes indicating the centroids of the microglial soma. These initial elements are all assigned a cost of zero, meaning that they have the highest priority. Each of these initial points become the root of a separate tree, since they each belong to a separate microglial cell. Once the priority queue has been initialized, we start using our modified version of Prim’s algorithm to construct arbors of the microglia. A detailed pseudo code description of this algorithm is presented in Appendix A. An overview of this procedure follows.

Each iteration of the graph construction algorithm consists of the following steps:

1. Remove node with the lowest cost from the queue. Steps 2 – 4 are only executed if it is a primary node that has not already been assigned to a tree.
2. Find the closest tree to node . Here ”closest” is meant in the sense of minimizing our cost metric. We will refer to the node closest to n that already belongs to as a leaf node.
3. Assign to tree all the nodes along the path from to the leaf node.
4. Update the cost of all the nodes along this path.
5. Search node neighborhood for other low-cost nodes and add them to the priority queue.

We repeat this procedure until the queue is empty.

Once a MST is generated for each microglial cell, interpolation is required to smooth the arbor representations. For this, all nodes that have been added to the MST are considered, with the exception of the root and leaf nodes. The location of each such intermediate node is updated to the average of its prior position, its parent position, and the position of its children. This averaging is weighted by the image intensity at each location, considering that brighter voxels are more likely to reside on arbors.

Finally, a pruning process is performed to reduce the “footprint” or artifacts of MSTs without losing important morphological details. In this pruning process, each node is marked as “active” or “inactive” before any pruning actually takes place. Initially, all nodes are marked inactive. A node is marked as active if any of the following conditions are met:

• The node is a branch point, having more than one child.

• The node is a leaf node, having no children.

• The node is a root node, having no parent.

• The node’s parent is inactive and its child is also inactive.

This prevents removal of consecutive nodes, which would otherwise result in loss of morphological detail. A node is explicitly marked as inactive if it is a leaf node whose distance from the branch point is less than an empirically set minimum offshoot length. In our experiments, this parameter was set to 15 voxels. This procedure results in removal of small branches that likely are tracing artifacts rather than genuine microglial arbor segments. Once all the nodes are marked, those marked as inactive are removed. Children of inactive parents are re-assigned to be children of their closest active ancestor. Finally, once pruning is complete, the MSTs are re-interpolated.

## Software Implementation Methods

The proposed arbor reconstruction algorithm is implemented in C++ using widely used open source libraries including ITK for common image processing steps (Ibanez *et al.*, 2003) and openMP for parallel processing (Dagum and Menon, 1998). We used Rubinstein’s KSVD library (Rubinstein *et al.*, 2010), and Zhang’s LC-KSVD library (Zhang and Li, 2010) in MATLAB for the dictionary learning steps. For representing the cell reconstructions, we used the widely used SWC open source file format (Cannon *et al.*, 1998). The design of our algorithm guarantees that the arbor reconstructions will be trees (no cycles).

The algorithms described above can be used for reconstructing individual 3-D image fields from the confocal microscope that fit into the computer’s memory. Analyzing the large mosaic fields like Figure 1 requires a more scalable implementation that is not limited by the computer memory. One idea in this regard is to trace the cells in the individual tiles, and then stitch together the resulting traces across the image boundaries. This task is extremely difficult, and a reliable method does not exist. With this in mind, we developed an indefinitely scalable “dice & trace” implementation that works as follows. The seamless mosaic image is divided into small “dices.” Each dice is a small 3-D image that is carved out from the mosaic, centered on one microglial cell, and just large enough to include the cell completely. Each dice may contain more than one cell, or parts thereof. The size of the dice can be easily specified, and for the current experiments it was set to. The size of the dice was chosen sufficiently large so that we don’t clip any arbors while carving the image. We run the multi-arbor reconstruction algorithm over each dice, but only retain the reconstruction for the cell in the center of the dice. These retained traces are then assembled together to form the reconstruction results for the large mosaic image. This approach has the disadvantage of performing repeated tracing of cells, since the reconstructed non-center cells in each dice are discarded. This is compensated adequately by the fact that dices can be processed in parallel by assigning one computing thread per dice, giving this approach the advantage of unlimited scalability. The total time to compute each dice is roughly 5-8 mins on a Dell PowerEdge server with Intel Xeon(R) 4870 2.4GHz processors with 500GB RAM, and this includes time for seed detection and tracing each dice. If this process was done using the traditional approaches then the time taken for the image with 3,300 cells would be 412.5 hours, however with our approach the time taken is 5.2 hours that is much better than the traditional approaches.

Appendix A

|  |
| --- |
| ***Algorithm 1*** *Detect Seed Points*  **Goal :** To obtain seed points given the dictionary and weights of classifier  **Step 1**: For each pixel in the image, extract the window centered at the pixel. (To reduce the computation cost, we have applied binarized mask and overestimated the foreground by reducing the threshold value).  **Step 2**: Get Sparse Features using OMP by solving the below equation  where , and . (4)  **Step 3** : Given the sparse codes and weights of classifier , classify the pixel as a potential seed point if  **Step 4**: Update the seed points list  **Step 5**: Filter out the seed points if they are not on the centerline of the arbor by retaining only those seed points, which are at the maximum intensity of profile in the window 15x15x3(same size as that of the patch). |

Algorithm. I. A pseudo-code summary of the proposed seed point detection algorithm.

|  |
| --- |
| ***Algorithm 2*** *Graph Construction Overview*  **while** is not empty do  ←  **if** **then**  **continue**  **end if**  **if** **then** {This node doesn’t belong to a tree yet}  *[, ] =*  **if** **then** {If a tree was found near }  *. =*  **while** is not empty **do**  {This is done so this node’s neighborhood will be searched}  **end while**  **end if**  **end if**  {More nodes may be added to the queue during this step}  **end while** |

Fig. I Pseudo code for tracing algorithm.

|  |
| --- |
| ***Algorithm 3*** *Find a tree near a given node*  *←*  *← 0*  *←{0, 0, 0} ← {}*  **while** *= 0 and* ***do*** *{500 iterations max}*  *{Find the direction where the cost decreases most rapidly}*  *←∇ Cost*  *{Simulate momentum by incorporating our previous direction}*  *←*  *{Find a new node by taking a step in this direction}*  *←*  *{Add this new node to our trail}*  **if****then** *{If the new node belongs to a tree}*  *←*  **else**  *←*  *←*  **end if**  **end while**  **if** *0* **then**  *return [, ]*  **end if** |

Fig. III Pseudo code to find the tree near a given node

|  |
| --- |
| **for** *all in n’s neighborhood* ***do***  *←*  *← {See Algorithm 4}*  **if****then**  *←*  **end if**  **end for** |

Fig. IV Pseudo code to find the low-cost neighbor near node n

|  |
| --- |
| ***Algorithm 4*** *Calculate the cost associated with adding pixel p to a tree.*  *: costs of the least-expensive neighbors in each cardinal direction*  ***Require:***  *← {pVal is the inverse of the intensity value of the input pixel p}*  *← 0*  *Δ ←*  **if** *Δ ≥ 0* **then**  *←*  **end if**  ***if******then***  *Δ ←*  **if** *Δ ≥ 0* **then**  *←*  **end if**  **if** **then**  *Cost ←*  **end if**  **end if**  **return** |

Fig. V Pseudo code to calculate the cost associated with adding a pixel p to a tree.