

Supplementary Material

Topology-based Sparsification of Graph Annotations

Daniel Danciu^{1,2,*} Mikhail Karasikov^{1,2,3,*} Harun Mustafa^{1,2,3}
André Kahles^{1,2,3,†} Gunnar Rättsch^{1,2,3,4,†}

¹Biomedical Informatics Group, Department of Computer Science, ETH Zurich, Zurich, Switzerland

²Biomedical Informatics Research, University Hospital Zurich, Zurich, Switzerland

³Swiss Institute of Bioinformatics, Zurich, Switzerland

⁴Department of Biology, ETH Zurich, Zurich, Switzerland

Algorithm S1 Backward traversal for anchor assignment

```
1: function TRAVERSEBWD(sink, visited[], anchors[], M)
2:   anchor[sink]  $\leftarrow$  True                                 $\triangleright$  mark the sink as anchor
3:   queue.push(sink, 0)                                        $\triangleright$  distance to next anchor is zero
4:   while not queue.empty() do
5:     node, depth  $\leftarrow$  queue.pop()
6:     if not visited[node] then                                 $\triangleright$  for detecting loops
7:       visited[node]  $\leftarrow$  True
8:       if depth = M then
9:         anchor[node]  $\leftarrow$  True
10:        depth  $\leftarrow$  0
11:      end if
12:      if Last(node) then                                        $\triangleright$  only for last outgoing nodes
13:        for all  $n \in$  Incoming(node) do
14:          queue.push(n, depth+1)                                $\triangleright$  go further from anchor
15:        end for
16:      end if
17:    end if
18:  end while
19: end function
```

*Joint-first authors.

†Joint corresponding authors; contact: andre.kahles@inf.ethz.ch and gunnar.ratsch@ratschlab.org.

Algorithm S2 Forward traversal for anchor assignment

```
1: function TRAVERSEFWD(node, visited[], anchor[], nearAnchor[], M)
2:   path  $\leftarrow$  []
3:   while not visited[node] do                                      $\triangleright$  Traverse until hitting a merge
4:     visited[node]  $\leftarrow$  True
5:     path.append(node)
6:     node  $\leftarrow$  lastOutgoing(node)
7:   end while
8:   for  $i = 0, \text{len}(\text{path}) - M, M$  do                              $\triangleright$  Assign anchor every M nodes
9:     anchor[path[i]]  $\leftarrow$  True
10:    nearAnchor[path[i:M]]  $\leftarrow$  True
11:   end for
12:   next  $\leftarrow$  lastOutgoing(path.back())
13:   if  $\text{len}(\text{path}) \bmod M == 0$  or nearAnchor[next] then
14:     return                                                          $\triangleright$  We merged close to an anchor, done
15:   end if
16:   anchor[path.back()]  $\leftarrow$  True
17:   nearAnchor[path[i:]]  $\leftarrow$  True
18: end function
```

Algorithm S3 Streaming construction of the pred and succ vectors

```
1: function CREATEPREDSUCC
2:   for start = 0, |V|, BLOCK_SIZE do                                $\triangleright$  traverse the graph in blocks of size BLOCK_SIZE
3:     parfor  $i = \text{start}, \text{start} + \text{BLOCK\_SIZE}$  do
4:       if a[i] = 0 then                                            $\triangleright$  Anchors don't have a successor
5:         succBuf[threadNo].append(lastSucc(i))
6:       end if
7:       for all j in pred(i) do
8:         if succ(j)=i then                                          $\triangleright$  Check if i is the RowDiff successor of j
9:           predBuf[threadNo].append(j)
10:          predBoundaryBuf.append(0)
11:         end if
12:       end for
13:       predBoundaryBuf.append(1)
14:     end parfor
15:      $\triangleright$  the following commands dump the memory buffer to disk
16:     succ.append(succBuf), pred.append(predBuf), predBoundary.append(predBoundaryBuf)
17:   end for
18: end function
```
