Additional File 1: MapReduce algorithm for similarity graph construction

In this algorithm, we have used the notation \(<\text{Key},\text{Value}>\) to denote the KeyValues in MapReduce and \(<\text{Key},[\text{Value}_1,\text{Value}_2,\ldots]\) to refer to the KeyMultiValues (shuffled KeyValues). For each Map and Reduce operation the input to the operation is shown inside parentheses in front of the name, and the output of the operation is separated by a colon ( : ).

Algorithm 1: Construction of the similarity graph in MapReduce

| Input: | \(S\): set of conserved regions, \(h\): current number of hash functions; \(h - d\): previous number of hash functions |
| Output: | \(G_h\): A similarity graph for conserved regions |

**Function**

\(CONSTRUCT_{\text{GRAPH}}_{\text{MAPREDUCE}}(S, h, h - d)\):

1. /* initialization: */
2. if This is the first call to the procedure then
3. Assign each conserved region to a processor;
4. Let \(S_{\text{proc}}\) denote the subset of \(S\) assigned to the processor \(\text{proc}\);
5. foreach conserved region \(S_i \in S_{\text{proc}}\) do
6. generate two empty tables \(t_{i1}\) and \(t_{i2}\) with zero columns and \(c(=2)\) rows. Each row \(r\) is to represent, respectively, the first- and second-level sketches generated by the \(r\)th hash function.
7. end
8. /* Main algorithm: */
9. /* Generating initial graph using first-level sketches */
10. Send each conserved region to its assigned processor;
11. Generate a new set of \(a, b,\) and \(p\) values;
12. foreach \(h_n = (h - d + 1)\ldots h\) do
13. foreach \(S_i \in S_{\text{proc}}\) with id \(S_{i,\text{id}}\) do
14. Compute a new sketch using the new hash function \(h_n\), and store it in a new row in the table \(t_{i1}\);
15. end
16. foreach sketch in \(t_{i1}\) do
17. Map(): emit \(<\text{sketch}, S_i,\text{id}>>\);
18. end
19. Shuffle;
20. Reduce (<sketch, \([S_i,\ldots]\) >):
21. \(<S_i, S_j>, <S_j, S_i>,\ldots>\);
22. Shuffle // sends KeyValue <\(S_i, S_j\)> to the processor designated to process \(S_i\).
23. /* Generating second step graph based on common neighbors */
24. foreach key \(S_i\) in the MapReduce do
25. \(\text{Neighbors}(S_i) := \text{values of } S_i;\)
26. \(\text{OldNeighbors}(S_i) := \text{retrieve the list of neighbors computed previously};\)
27. \(\text{NewNeighbors}(S_i) := \text{Neighbors}(S_i) - \text{OldNeighbors}(S_i);\)
28. foreach \(h_n = (h - d + 1)\ldots h\) do
29. Apply hash function \(h_n\) on \(\text{NewNeighbors}(S_i)\)
30. and update the previous computed sketches (row \(h_n\) on \(t_{i2}\));
31. end
32. foreach \(h_n = 1\ldots h\) do
33. Apply hash function \(h_n\) on \(\text{Neighbors}(S_i)\) and compute new sketches;
34. end
35. foreach sketch in \(t_{i2}\) do
36. Map(): emit \(<\text{sketch}, S_{i,\text{id}}>>\);
37. end
38. Shuffle;
39. Reduce (<sketch, \([S_i, S_j,\ldots]\) >):
40. \(<S_i, S_j>, <S_j, S_i>,\ldots>\);
41. foreach KeyValue <\(S_i, S_j\)> do
42. add an edge between the nodes \(S_i\) and \(S_j\) (if the edge does not exist);
43. end