```
numberofRBN = 2000; (*number of RBN's with the same topology*)
numberofexps = 50; (*number of topologies to test*)
numperms = 5000; (*random permuations of the updating functions to test*)
n = 5; (*number of nodes*) k = 3;  (*in-degree*)

complexityRBNtrimmed = Table[0, numberofexps];
complexityBoolFun = Table[0, numberofexps];
complexityRBN = Table[0, numberofRBN];

    (*generate the updating functions*)
Do[booleanfunction = {}; booleanfunctionMatrix = {}; Clear[a];
 A = Table[a[i], {i, 1, k}];
 For[i = 0, i < n, i++,
  f = BooleanFunction[RandomInteger[{0, (2^2^k - 1)}], k];
  AppendTo[booleanfunction, f];
  AppendTo[booleanfunctionsMatrixList,
    Boole[BooleanTable[BooleanConvert[Apply[f, A], "NOR"], A]]] ×
   AppendTo[booleanfunctionMatrix, BooleanConvert[Apply[f, A], "NOR"]] ];

 r = 0; (*generate the topology*)
 Do[r++; iterations = 1;
  Mnodes = Table[Table[0, n], n];
  For[i = 1, i ≤ n, i++, Flag = 0;
   While[Flag < k, position = RandomInteger[{1, n}];
    If[Mnodes[[i, position]] == 0, Mnodes[[i, position]] = 1; Flag++]]];
  Matrixnodes = Transpose[Mnodes];

      (*generate the dynamics of the network*)
  inputstates = Tuples[{0, 1}, n]; (*possible input states*)
  outputstates = Table[Table[0, iterations], 2^n]; (*output states*)
  For[q = 1, q ≤ 2^n, q++,
   neighbors = Table[0, k]; (*positions of the k-neighbors*)
   For[i = 1, i ≤ iterations, i++,
    statesnodes = Table[0, n]; (*states of the nodes*)
    For[j = 1, j ≤ n, j++ , (*run over the nodes*) Flag = 0;
     For[p = 1, p ≤ n, p++, (*find the k-neighbors*)
      If[Matrixnodes[[p, j]] == 1, Flag++;
       neighbors[[Flag]] = p]];
     For[m = 1, m ≤ k , m++,
      (*run over the in-degrees*) a[m] = inputstates[[q, neighbors[[m]]]]];
     statesnodes[[j]] = FullSimplify[booleanfunctionMatrix[[j]]]];
    (*transform the states of the nodes into the state of the network*)
    outputstates[[q]] = FullSimplify[Boole[statesnodes]];
    inputstates[[q]] = FullSimplify[Boole[statesnodes]]]];

      (*measure the complexity of the RBN*)
  If[First[Flatten[outputstates]] == 0,
   codedynamics = "0" <> ToString[FromDigits[Flatten[outputstates]]],
   codedynamics = ToString[FromDigits[Flatten[outputstates]]]];
  complexityRBN[[r]] = StringBDM[codedynamics], numberofRBN];

     (*measure the complexity of the updating functions*)
 rg = booleanfunctionsMatrixList; mat = Flatten[rg];
 perms = RandomChoice[Permutations[Table[i, {i, n}]], numperms];
 isos = DeleteDuplicates[Permute[rg, #] & /@ perms] ;
 complexityiso = Table[0, Length[isos]];
 Do[flatten = Flatten[isos[[L]]];
  If[First[flatten] == 0,
   output = "0" <> ToString[FromDigits[flatten]], output = ToString[FromDigits[flatten]]];
  complexityiso[[L]] = StringBDM[output], {L, Length[isos]}];
 complexityBoolFun[[y]] = Min[complexityiso];

     (*mean complexity of the RBN's*)
 complexityRBNtrimmed[[y]] = TrimmedMean[complexityRBN,
   {(Length[Select[complexityRBN, # ≤ Quantile[complexityRBN, 1/4] &]]) / Length[complexityRBN],
    (Length[Select[complexityRBN, # ≥ Quantile[complexityRBN, 3/4] &]]) /
     Length[complexityRBN]}], {y, numberofexps}]
Export[NotebookDirectory[] <> "C(f)_min.txt", complexityBoolFun];
Export[NotebookDirectory[] <> "C(BN)_trim.txt", complexityRBNtrimmed];
```