Ecography

E7281
Nilsson Jacobi, M., André, C., Döös, K. and Jonsson, P. R. 2012. Identification of subpopulations from connectivity matrices. – Ecography 35: xxx–xxx.

Supplementary material

Appendix 1

- **Import the connectivity matrix. The file path has to be modified.**

```
p = Import[PATH <> "matrix", "MatrixMarket"];
coord = Import[PATH <> "Coordinates.csv", "CSV"];
ListPlot[coord, PlotRange → All, Axes → None, Frame → True]
MatrixPlot[p]
```

```
pp = (p + Transpose[p]) / 2.0;
dia = SparseArray[
   DiagonalMatrix[Table[1 / (Total[pp[[i]]] + 10^-10), {i, Length[pp]}]]];
pp = pp.dia; (*this is to normalize all rows,
it is optional and depends on your desired definition of leakage*)
pp = (pp + Transpose[pp]) / 2.0;
Do[pp[[i, i]] = 0.0, {i, Length[pp]}];
```

- **The slit function takes a list of sites and returns two lists. It also accepts an input parameter defining how many times to try the spilt (the best result from the tries is returned).**

```
split[index_, tries_] := {
   ppp = pp[[index, index]]; (*makes a submatrix of the total
    connectivity matrix only involving the sites in the index list *)
   th = 10^-10;
   alpha = 0.1;
   maxIt = 500;
   n = Length[ppp];
   best = 10^10;
   s = Table[1.0, {n}];
   eNoSplit = -s.ppp.s + al * Total[s]^2;

   Do[
    s = RandomReal[{-1.0, 1.0}, n];
    ds = s + 1;
    (*ds is set different from s to ensure that the While loop starts*)
    sTot = Total[s];
    it = 0;
    sOld = Sign[s];
    While[And[Norm[s - ds] > th, it < maxIt],
     it++;
     v = ppp.s - al * Total[s];
     (* the following three lines implements EQ. 8 in the paper*)
     ds = Abs[v]^(1 / 2) * Sign[v];
     s = alpha * s + (1 - alpha) * ds;
    ];
    If[it > maxIt, Print["failed"]];
    s = Sign[s];
    e = -s.ppp.s + al * Total[s]^2;
    (* calclulates the value of the cost function*)
    If[e < best,
     sBest = s;
     best = e];
    , {tries}];

   s = sBest;
   part = {};
   Do[
    If[s[[i]] == -1, AppendTo[part, index[[i]]]];
    , {i, n}];

   If[And[Length[part] > 0, Length[part] < n, best < eNoSplit],
    notPart = Complement[index, part];
    return = {part, notPart},
    return = {index}];
   return
  }[[1]]
```

- **This funtion splits the index list recursively until non of the subpopulations can be split further to improve the minimization**

```
recSplit[ta_, tries_] := {
   old = 0;
   ret = ta;
   While[Length[ret] > old,
    old = Length[ret];
    ret = Flatten[Table[split[ret[[i]], tries], {i, Length[ret]}], 1];
   ];
   ret}[[1]]
```

- **This function tries to merge random subopoulations, checkin if the result is a better soluton to the minimization problem**

```
merge[ta_] := {
   nIt = Length[ta]^2;
   ret = ta;
   Do[
    i = RandomInteger[{1, Length[ret]}];
    j = RandomInteger[{1, Length[ret]}];
    If[i ≠ j,
     li = Flatten[{ret[[i]], ret[[j]]}];
     pTest = pp[[li, li]];

     s = Table[1.0, {Length[li]}];
     eTogether = -s.pTest.s + al * Total[s]^2;

     s[[1 ;; Length[ret[[i]]]]] = -1.0;
     eSplit = -s.pTest.s + al * Total[s]^2;

     If[eTogether < eSplit,
      ret[[i]] = li;
      ret = Drop[ret, {j}];
     ];
    ];
    , {it, nIt}];
   ret}[[1]]
```

- **The quality function is not used to evaluate the result of the solution, i.e. it measures the average total leakage between the subpopulations**

```
qual[ta_] := {
   pi = Table[0.0, {Length[p]}, {Length[ta]}];
   Do[
    Do[
      pi[[ta[[i, j]], i]] = 1.0;
      , {j, Length[ta[[i]]]}];
    , {i, Length[ta]}];
   pt = Inverse[Transpose[pi].pi].Transpose[pi].Transpose[p].pi;
   (* pt is the reduced cinnectivity matrix, see Table 1 *)
   ptt = pt;
   Do[
     If[Total[ptt[[i]]] > 0,
       ptt[[i]] = ptt[[i]] / Total[ptt[[i]]]];
     , {i, Length[ptt]}]
    Do[
     ptt[[i, i]] = 0.0;
     , {i, Length[ptt]}];
   Mean[Total[ptt]]}[[1]]
```

- **This is an example of how the functions can be combined to give a solution to the minimization problem for a specific value of beta**

```
beta = Length[p];
al = 1 / beta;
ta = {Range[1, Length[pp]]};
taOld = {};
While[ta ≠ taOld,
  taOld = ta;
  ta = recSplit[ta, 5];
  Print["Splitting into ", Length[ta], " clusters"];
  ta = merge[ta];
  Print["Merging into ", Length[ta], " clusters"];
 ];

ListPlot[Table[coord[[ta[[i]]]], {i, Length[ta]}], Frame → True, Axes → False]
qual[ta]
```

- **This is the total algorithm where we cycle through beta to produce different levels of coarse gaining of the partitioning of the population**

```
cycles = 2;
steps = 10;
res = {};
Do[
  ta = {Range[1, Length[pp]]};
  Do[
   beta = Length[p] / (1 + 0.8 Sin[t])^3;
   (* a periodic function that cycles through the beta values,
   details are not important, should start at 1 though *)
   al = 1 / beta;
   taOld = {};
   While[ta ≠ taOld,
    taOld = ta;
    ta = recSplit[ta, 5];
    ta = merge[ta];
   ];
   q = qual[ta];
   Print[itG, "  ", al * Length[p], "    ", Length[ta], "    ", q];
   AppendTo[res, {al * Length[p], Length[ta], q, ta}];
   , {t, 0, 3 Pi / 2, Pi / steps}];
  , {itG, cycles}];
```