Social Networks in Biology

File: SocNetSampleText.nb

To accompany
"Getting the 'Edge' on the Next Flu Pandemic: We Should'a 'Node' Better"
By Angela B.Shiflet and George W.Shiflet
Wofford College, Spartanburg, South Carolina

- **This file deals with four people from "activities-portland-1-v1.txt" at http://ndssl.vbi.vt.edu/opendata/download.php and uses all their activities.**

Based on
Eubank, S., V.S. Anil Kumar, M. Marathe, A. Srinivasan and N. Wang. 2004. "Structural and Algorithmic Aspects of Large Social Networks." Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 711-720.

Data downloaded from
http://ndssl.vbi.vt.edu/opendata/download.php

NDSSL (Network Dynamics and Simulation Science Laboratory, Virginia Polytechnic Institute and State University). 2009. "NDSSL Proto-Entities" http://ndssl.vbi.vt.edu/opendata/ Accessed 8/27/9.
_____. 2009. Synthetic Data Products for Societal Infrastructures and Proto-Populations: Data Set 1.0. ndssl.vbi.vt.edu/Publications/ndssl-tr-06-006.pdf
_____. 2009. Synthetic Data Products for Societal Infrastructures and Proto-Populations: Data Set 2.0. ndssl.vbi.vt.edu/Publications/ndssl-tr-07-003.pdf
_____. 2009. Synthetic Data Products for Societal Infrastructures and Proto-Populations: Data Set 3.0. ndssl.vbi.vt.edu/Publications/ndssl-tr-07-010.pdf

Connection matrix

- **get list of locations**

```
Clear[genLocIDLst];
genLocIDLst[activities_] := Module[{locs},
  locs = Transpose[activities][[7]];
  DeleteDuplicates[locs]
 ]
```

```
activities = {{4, 7, 1, 0, 0, 13950, 2938},
   {4, 7, 2, 1, 27000, 17550, 27618}, {4, 7, 3, 4, 64800, 29250, 2938},
   {4, 8, 4, 0, 0, 17400, 2938}, {4, 8, 5, 2, 34200, 3060, 6270},
   {4, 8, 6, 2, 38400, 3060, 21032}, {4, 8, 7, 0, 43200, 3150, 2938},
   {4, 8, 8, 2, 47099, 4215, 15370}, {4, 8, 9, 4, 54299, 34500, 2938},
   {5, 9, 1, 0, 0, 13350, 10628}, {5, 9, 2, 1, 25800, 19049, 29740},
   {5, 9, 3, 4, 61800, 30749, 10628}, {6, 18, 25, 0, 0, 15300, 2938},
   {6, 18, 26, 7, 30600, 18149, 5212}, {6, 18, 27, 4, 65700, 1800, 2938},
   {6, 18, 28, 2, 67860, 2388, 19815}, {6, 18, 29, 0, 70499, 26400, 2938}};
```

```
personIdLst = {7, 8, 9, 18};
locationIDLst = genLocIDLst[activities]
```

```
{2938, 27 618, 6270, 21 032, 15 370, 10 628, 29 740, 5212, 19 815}
```

- **plot of graph**

```
GraphPlot[{7 → 2938, 7 → 27 618, 8 → 2938, 8 → 6270, 8 → 21 032, 8 → 15 370,
  9 → 10 628, 9 → 29 740, 18 → 2938, 18 → 5212, 18 → 19 815}, VertexLabeling → True,
 VertexCoordinateRules → {7 → {-1, 7.5}, 8 → {-1, 5.5}, 9 → {-1, 2.5}, 18 → {-1, 0.5},
   2938 → {4, 8}, 27 618 → {4, 7}, 6270 → {4, 6}, 21 032 → {4, 5},
   15 370 → {4, 4}, 10 628 → {4, 3}, 29 740 → {4, 2}, 5212 → {4, 1}, 19 815 → {4, 0}},
 VertexRenderingFunction → ({White, EdgeForm[Black], Disk[#, .4], Black, Text[#2, #1]} &)]
```

- **function to return index of element in a list**

```
Clear[index];
index[el_, lst_] := Flatten[Position[lst, el]][[1]]
```

```
(* test *)
index[15 370, locationIDLst]
```

```
5
```

- **generate connection matrix with people indices as row labels and location indices as column labels**
  Function to generate people-to-location graph

```
Clear[genPeopleLocConnMat];
genPeopleLocConnMat[people_, locs_, activities_] := Module[{connMat},
  connMat = Table[0, {Length[people]}, {Length[locs]}];
  Do[connMat[[index[activities[[i, 2]], people], index[activities[[i, 7]] , locs]]] = 1,
   {i, Length[activities]}];
  connMat
 ]
```

```
connMat = genPeopleLocConnMat[personIdLst, locationIDLst, activities];
```

```
TableForm[connMat]
```

1	1	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0
0	0	0	0	0	1	1	0	0
1	0	0	0	0	0	0	1	1

```
Length[Transpose[connMat]]
```

```
9
```

Minimum dominating set problem

- **function to return the degree of a location node**

```
Clear[degLocation];
degLocation[connMat_, j_] := Count[Transpose[connMat][[j]], 1]
```

- **list of ordered pairs of location index & corresponding degree**

```
locDegPairLst = Table[{j, degLocation[connMat, j]}, {j, Length[Transpose[connMat]]}]
```

```
{{1, 3}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1}}
```

- **function to return *lst* sorted by the second members of the ordered pairs**

```
Clear[sortSecond];
sortSecond[lst_] := Sort[lst, #1[[2]] > #2[[2]] &]
```

```
(* test*)
sortSecond[locDegPairLst]
```

```
{{1, 3}, {9, 1}, {8, 1}, {7, 1}, {6, 1}, {5, 1}, {4, 1}, {3, 1}, {2, 1}}
```

- **Function to return list of indices of personIDs adjacent to location with index *j***

```
Clear[adjacentPeopleLst];
adjacentPeopleLst[connMat_, j_] := Flatten[Position[Transpose[connMat][[j]], 1]]
```

```
(* test *)
adjacentPeopleLst[connMat, 1]
```

```
{1, 2, 4}
```

```
(* test *)
adjacentPeopleLst[connMat, 9]
```

```
{4}
```

- **function to return partial minimum dominating set to cover percent fraction of the people using FastGreedy Algorithm**

```
Clear[minDominating];
minDominating[locationIDLst_, connMat_, percentPeople_] :=
 Module[{people, locations, locDegPairLst, sortedLocDegPairLst,
    locDegPair, locIndex, locDeg, loc, percentLength, j},
  If[percentPeople < 0 || percentPeople > 1, percentPeople = 1];
  people = {};
  locations = {};
locDegPairLst = Table[{j, degLocation[connMat, j]}, {j, Length[Transpose[connMat]]}];
  sortedLocDegPairLst = sortSecond[locDegPairLst];
  locDegPair = 1;
  percentLength = percentPeople * Length[connMat];
While[Length[people] < percentLength,
 {locIndex, locDeg} = sortedLocDegPairLst[[locDegPair]];
 loc = locationIDLst[[locIndex]];
 locations = Union[locations, {loc}];
 people = Union[people, adjacentPeopleLst[connMat, locIndex]];
 locDegPair++
   ];
  locations
]
```

```
(* test *)
locations = minDominating[locationIDLst, connMat, 1]
```

```
{2938, 5212, 19815, 29740}
```

```
Length[locations]
```

```
4
```

```
(* test *)
locations = minDominating[locationIDLst, connMat, 0.5]
```

```
{2938}
```

```
Length[locations]
```

```
1
```

```
(* test *)
locations = minDominating[locationIDLst, connMat, 0.75]
```

```
{2938}
```

People-to-people graph and degree distribution

- **function to generate connection matrix for a people-to-people graph**

```
Clear[peopleToPeople];
peopleToPeople[connMat_] := Module[{maxPersonIndex, connPeopleMat, i, locIndex, j},
  maxPersonIndex = Length[connMat];
  connPeopleMat = Table[0, {maxPersonIndex}, {maxPersonIndex}];
  (* go through every column of connMat *)
  Do[
   (* go down locIndex column looking for 1's *)
   Do[
    If[connMat[[i, locIndex]] == 1,
     (* for every 1 look through rest of locIndex column looking for 1's *)
     (* These people are adjacent *)
     Do[
      If[connMat[[j, locIndex]] == 1,
       connPeopleMat[[i, j]] = connPeopleMat[[j, i]] = 1], {j, i + 1, maxPersonIndex}
     ]
    ],
    {i, maxPersonIndex}
   ],
   {locIndex, Length[Transpose[connMat]]}
  ];
  connPeopleMat
 ]
```

```
connPeopleMat = peopleToPeople[connMat];
```

```
TableForm[connPeopleMat]
```

0	1	0	1
1	0	0	1
0	0	0	0
1	1	0	0

- **graph**

```
GraphPlot[connPeopleMat, VertexLabeling → True]
```

```
GraphPlot[{7 → 8, 7 → 18, 8 → 18, 9 → 9}, VertexLabeling → True, VertexCoordinateRules →
  {7 → {-1, 0}, 8 → {0, 1}, 9 → {1.2, 1}, 18 → {2, 0}}, SelfLoopStyle → 0.01,
 VertexRenderingFunction → ({White, EdgeForm[Black], Disk[#, .15], Black, Text[#2, #1]} &)]
```

- **degree distribution of people-to-people graph**

- **function to return the degree of a person node in people-to-people graph**

```
Clear[degPersonPPG];
degPersonPPG[connPeopleMat_, i_] := Count[connPeopleMat[[i]], 1]
```

- **list, *distribLst*, of degrees of each vertex**

```
distribLst = Table[degPersonPPG[connPeopleMat, i], {i, Length[connPeopleMat]}]
```

```
{2, 2, 0, 2}
```

- **function to return the degree distribution list: If n is the number of nodes in the network and nk is the number of nodes of degree k, then the degree distribution is P(k) = nk/n, which is the proportion of nodes having degree k.**

```
Clear[pLst];
pLst[connPeopleMat_] := Module[{numPeople, degreeLst, i, deg},
  numPeople = Length[connPeopleMat];
  degreeLst = Table[degPersonPPG[connPeopleMat, i], {i, numPeople}];
  Table[{deg, Count[degreeLst, deg] / numPeople}, {deg, 0, Max[degreeLst]}]
 ]
```

```
lst = pLst[connPeopleMat]
```

$\left\{\left\{0, \frac{1}{4}\right\}, \{1, 0\}, \left\{2, \frac{3}{4}\right\}\right\}$

```
ListPlot[lst, Ticks → {{0, 1, 2}, {.25, .50, .75, 1.00}}, PlotStyle → PointSize[0.025],
 PlotRange → {{-0.05, 2.1}, {-0.05, 1}}, AxesLabel → {"Degree" k, P[k]}]
```

- **average degree**

```
tbl = Table[degPersonPPG[connPeopleMat, i], {i, 4}]
Mean[tbl] // N
```

```
{2, 2, 0, 2}
```

```
1.5
```

Clustering coeff in people-to-people graph

- **Function to return list of indices of those adjacent to person with index i in person-to-person graph**

```
Clear[adjacentPeople];
adjacentPeople[connPeopleMat_, i_] :=
 Flatten[Position[connPeopleMat[[i]], 1]]
```

```
(* test *)
TableForm[Table[{i, adjacentPeople[connPeopleMat, i]}, {i, Length[connPeopleMat]}] ]
```

```
1    2
     4
2    1
     4
3
4    1
     2
```

- **Function to return the number of edges in a set in person-to-person graph**

```
Clear[numPeopleEdges];
numPeopleEdges[connPeopleMat_, vertices_] := Module[{subMat, trans},
  subMat = connPeopleMat[[vertices]];
  trans = Transpose[subMat][[vertices]];
  Count[trans, 1, 2] / 2
 ]
```

```
(* test *)
numPeopleEdges[connPeopleMat, {2, 4, 1}]
```

```
3
```

- **function to return the clustering coefficient for a node**
  For a node with 0 or 1 adjacent nodes, return 1

```
Clear[clusteringCoeff];
clusteringCoeff[connPeopleMat_, v_] := Module[{deg, adj, numerator, denominator},
  deg = degPersonPPG[connPeopleMat, v];
  If[deg < 2, 0,
   adj = adjacentPeople[connPeopleMat, v];
   numerator = numPeopleEdges[connPeopleMat, adj];
   denominator = deg * (deg - 1) / 2.0;  (* floating point number of combinations *)
   numerator / denominator
  ]
 ]
```

```
(* test *)
clusteringCoeff[connPeopleMat, 1]
```

```
1.
```

```
(* test *)
clusteringCoeff[connPeopleMat, 3]
```

```
0
```

- **average clustering coefficient**

```
Clear[meanClusteringCoeff]
meanClusteringCoeff[connPeopleMat_] :=
 Mean[Table[clusteringCoeff[connPeopleMat, v], {v, Length[connPeopleMat]}]]
```

```
(* test *)
meanClusteringCoeff[connPeopleMat]
```

```
0.75
```