MATHEMATICA ROUTINES FOR INDEX EXPECTATION AND PERCOLATION

OLIVER KNILL

ABSTRACT. The following Mathematica routines first give an index algorithm to compute the Euler characteristic of finite graphs, and a routine to explore that the index expectation for injective maps is equal to curvature. In the proof of that index expectation result, we also prove a network stability theorem which can be illustrated with routines provided here.

1. EULER CHARACTERISTIC COMPUTATION

Computing the Euler characteristic $\chi(G) = \sum_{k=0}^{\infty}(-1)^k v_k$ of a simple graph $G = (V, E)$ by counting the number v_k of complete graphs K_{k+1} embedded into G is inefficient and quickly hits a wall because counting cliques in a graph is a computationally hard problem. We have introduced an index $i_f(x) = 1 - \chi(S_f^-(x))$ and a Poincaré Hopf theorem $\chi(G) = \sum_{x \in V} i_f(x)$ which allows much faster computation. The reason is that the graph $S_f^- = \{y \in S(x) \mid f(y) < f(x)\}$ is in general small even so the sphere $S(x)$ in a graph does not need to be small. We can expect that to compute the Euler characteristic of a graph of order n we have to add n Euler characteristics computations of graphs of order $n/2$.

The first routine ErdoesRenyi in the following block generates a random graph with M vertices, where each of the possible links is turned on with probability p. The routine EulerChi computes the Euler characteristic of a graph. It is recursive and calls itself.

```
ErdoesRenyi[M_,p_]:=Module[{q,e,a},V=Range[M];
  e=EdgeRules[CompleteGraph[M]]; q={};
  Do[If[Random[]<p,q=Append[q,e[[j]]]],{j,Length[e]}];
  UndirectedGraph[Graph[V,q]]];
UnitSphere[s_,a_]:=Module[{b=NeighborhoodGraph[s,a]},
  If[Length[VertexList[b]]<2,Graph[{}],VertexDelete[b,a]]];
EulerChi[s_]:=Module[{vl,n,sp,u,g,sm,ff,a,k,el,m,q},
  vl=VertexList[s]; n=Length[vl];
  ff=Range[n]; el=EdgeList[s]; m=Length[el];
  g[b_]:=ff[[Position[vl,b][[1,1]]]];
  If[n==0,0,If[n==1 || m==Binomial[n,2],1,If[m==0,n,
  u=Table[ A=g[vl[[a]]]; sp=UnitSphere[s,vl[[a]]];
```

Date: February 20, 2012.
1991 *Mathematics Subject Classification.* Primary: 05C80,05C82,05C10,90B15,57M15 .
Key words and phrases. Random Graph Theory, Complex Networks, Graph Dimension, Graph Curvature, Euler characteristic.

```
  q=VertexList[sp]; sm={};
  Do[If[g[q[[k]]]<A,sm=Append[sm,q[[k]]]],{k,Length[q]}];
   If[Length[sm]==0,1,(1-EulerChi[Subgraph[sp,sm]])],{a,n}];
  Sum[u[[k]],{k,n}]]]]]];
ss=ErdoesRenyi[30,0.4]; EulerChi[ss]
```

These routines could be optimimized by choosing functions f cleverly. It appears in that in the Erdoes-Renyi probability space $E(n,p)$ most graphs allow a computation of the Euler characteristic in polynomial time.

2. Site percolation

If we take an arbitrary host graph G and a graph H and knock of each vertex with probability p. What fraction of patterns H survive the attack? If we integrate this over p from 0 to 1, then we get a universal average $1/(\text{ord}(H)+1)$, where $\text{ord}(H)$ is the order of the graph. In the case of triangles, the decimation factor is $1/4$. The following routines compute this numerically. We chose first an Erdoes-Renyi graph with 20 vertices where each edge is taken with probability $1/2$. Now, we make 70 Monte Carlo experiments and sum up over 30 different p values. We have used the site percolation result in [1].

```
NumberOfCliques[K_,k_]:=Module[
  {n,m=Length[EdgeList[K]],s,u,V=VertexList[K],U},
  s=Subsets[V,{k,k}]; n=Length[V]; If[k==1,u=n,If[k==2,u=m,
  u=Sum[If[Length[EdgeList[Subgraph[K,s[[j]]]]]==
       Binomial[k,2],1,0],{j,Length[s]}]]];u];
G1=ErdoesRenyi[20,1/2]; V1=VertexList[G1];
v1=NumberOfCliques[G1,3];
F[p_]:=Module[{q=0,U,G2},Do[G2=G1;
  Do[If[Random[]<p && Length[VertexList[G2]]>1,
  G2=VertexDelete[G2,V1[[k]]]],{k,Length[V1]}];
   If[(Length[VertexList[G2]]<=1 ||
       Length[EdgeList[G2]]<3),v2=0,
  v2=NumberOfCliques[G2,3]];
  q+=(v2/v1),{70}]; N[q/70]];
Sum[F[l/30],{l,0,30}]/31
```

3. Bond percolation

If we take an arbitrary host graph G and a graph H and knock of each edge with probability p. What fraction of patterns H survive the attack? If we integrate this over p from 0 to 1, then we get a universal average $1/(\text{size}(H)+1)$, where $\text{size}(H)$ is the size of the graph. In the case of tetrahedra, the decimation factor is $1/7 \sim 0.143$. The following routines compute this numerically. We chose first an Erdoes-Renyi graph with 12 vertices where each edge is taken with probability $1/2$. Now, we make 70 Monte Carlo experiments and sum up over 30 different p values.

```
G1=ErdoesRenyi[12,1/2]; V1=VertexList[G1];
  v1=NumberOfCliques[G1,4]; E1=EdgeList[G1];
F[p_]:=Module[{q=0,U,G2},Do[G2=G1;E2=EdgeList[G2];
  Do[If[Random[]<p,E2=Complement[E2,{E1[[k]]}]];
```

```
  G2=Graph[V1,E2]],{k,Length[E1]}];
  If[Length[EdgeList[G2]]<4,v2=0,v2=NumberOfCliques[G2,4]];
  q+=(v2/v1),{70}]; N[q/70]];
Sum[F[1/30],{l,0,30}]/31
```

The numbers have been chosen small so that the wait is not too long.

4. Index expectation

The next routines illustrate the theorem that the average index $i_f(x)$ of a graph is equal to curvature $K(x)$ [1]. This result links Gauss-Bonnet $\sum_{x \in V} K(x) = \chi(G)$ and Poincaré-Hopf $\sum_{x \in V} i_f(x) = \chi(G)$ which both hold for arbitrary finite simple graphs. Just take the expectation of Poincaré-Hopf to get Gauss-Bonnet. Here are the base routines, which allow to compute index and curvature.

```
Curvature[s_,v_]:=Module[{s1,k,r},s1=UnitSphere[s,v];
  vl1=VertexList[s1]; n1=Length[vl1];
  r=Table[(-1)^k/(k+1),{k,n1}];
  u=Table[NumberOfCliques[s1,k],{k,n1}]; 1+u.r];
Curvatures[s_]:=Module[{},vl=VertexList[s];
  Table[Curvature[s,vl[[k]]],{k,Length[vl]}]]
index[f_,s_,a_]:=Module[q{},vl=VertexList[s];
  sp=UnitSphere[s,a]; q=VertexList[sp]; sminus={};
   Do[If[f[[Position[vl,q[k]]][[1 ,1]]]]
       -f[[Position[vl,a][[1,1]]]] <0 ,
      sminus=Append[sminus,q[[k]]]],{k,Length[q]}];
   1-EulerChi[Subgraph[s,sminus]]];
```

Here is an experiment with small parameters. Adapt the number of Monte-Carlo experiments to get closer to a match:

```
G1=ErdoesRenyi[12,1/2]; V1=VertexList[G1];
morsefunction:=Table[2 Random[]-1,{Length[V1]}];
IndexList[f_,s_]:=Module[{},vl=VertexList[s];n=Length[vl];
  Table[index[f,s,vl[[k]]],{k,Length[vl]}]];
curvaturelist=Curvatures[G1];
Sum[IndexList[morsefunction,G1],{100}]/100-curvaturelist
```

References

[1] O. Knill. On index expectation and curvature for networks. http://arxiv.org/abs/1202.4514, 2012.

E-mail address: knill@math.harvard.edu

Department of Mathematics, Harvard University, Cambridge, MA, 02138