graph_connectivity

Robin Deits

February 17, 2014

In [45]:
```
using PyPlot
```

```
Loading help data...

Warning: Possible conflict in library symbol dtrtri_
Warning: Possible conflict in library symbol dgetri_
Warning: Possible conflict in library symbol dgetrf_
```

In [114]:
```
module Con
export Graph, Node, Edge

type Node
    label::ASCIIString
end

type Edge
    a::Node
    b::Node
end

type Graph
    nodes::Set{Node}
    neighbors::Dict{Node,Set{Node}}
end

function Graph(nodes::Set{Node}, edges::Array{Edge})
    neighbors = Dict{Node,Set{Node}}();
    for e in edges
        for (n1, n2) in ((e.a, e.b), (e.b, e.a));
            for e in edges
                if haskey(neighbors, n1)
                    push!(neighbors[n1], n2);
                else
                    neighbors[n1] = Set(n2);
                end
            end
        end
    end
    for n in nodes
        if !haskey(neighbors, n)
            neighbors[n] = Set{Node}()
        end
    end
    Graph(nodes, neighbors)
end

type Path
    nodes::Array{Node}
```

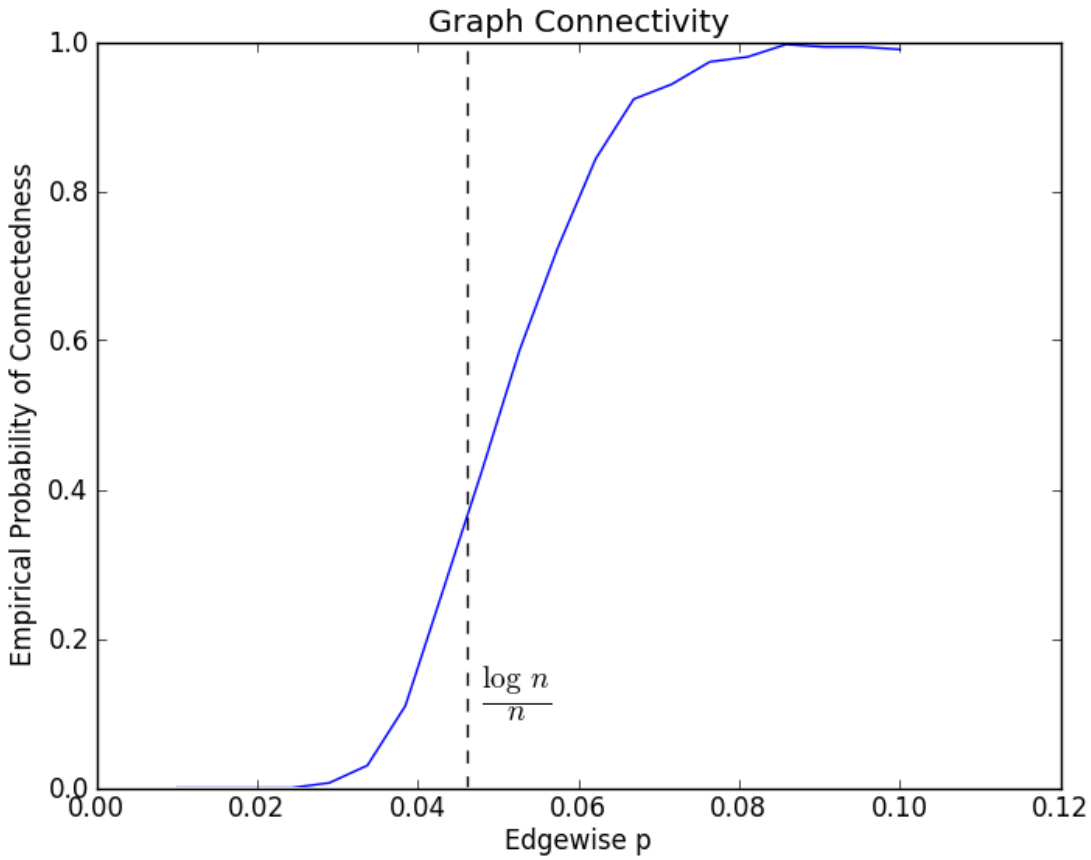
```
end

function random_graph(n::Int64, p::Float64)
    nodes = [Node(string(x)) for x in 1:n]
    edges = Edge[]
    for (i, n1) in enumerate(nodes)
        for n2 in nodes[i+1:end]
            if rand() <= p
                push!(edges, Edge(n1, n2))
            end
        end
    end
    return Graph(Set(nodes...), edges)
end

function BFS_path(graph::Graph, s::Node, t::Node)
    visited = Set(s)
    active_set = [Path([s])]
    while true
        new_active_set = Path[]
        for p in active_set
            for u in graph.neighbors[p.nodes[end]]
                n = copy(p.nodes)
                push!(n, u)
                new_path = Path(n)
                if u == t
                    return new_path
                end
                if !(u in visited)
                    push!(visited, u)
                    push!(new_active_set, new_path)
                end
            end
        end
        if length(new_active_set) == 0
            return Path(Node[])
        end
        active_set = new_active_set
    end
end

function is_fully_connected(graph::Graph)
    s, state = next(graph.nodes, start(graph.nodes))
    visited = Set{Node}(s)
    active_set = Node[s]
    while true
        new_active_set = Node[]
        for n in active_set
            for u in graph.neighbors[n]
                if !(u in visited)
                    push!(visited, u)
                    push!(new_active_set, u)
                    if length(visited) == length(graph.nodes)
                        return true
                    end
                end
            end
        end
        if length(new_active_set) == 0
            return false
        end
        active_set = new_active_set
    end
end

end
```

```
Warning: replacing module Con
```

In [115]:
```
import Con
```

In [162]:
```
conn_freqs = Float64[]
ps = linspace(0.01, 0.1, 20)
tries = 300
n = 100
for p in ps
    count = 0
    for j = 1:tries
        g = Con.random_graph(n, p)
        if Con.is_fully_connected(g)
            count += 1
        end
    end
    push!(conn_freqs, count / tries)
end
```

In [187]:
```
plot(ps, conn_freqs)
xlabel("Edgewise p")
ylabel("Empirical Probability of Connectedness")
title("Graph Connectivity")
axvline(log(n)/n; color="k", linestyle="--")
annotate("\$\\frac{\\log\\, n}{n}\$", (0.048, 0.1), fontsize=20)
```



Out [187]:

```
PyObject <matplotlib.text.Annotation object at 0x11e0b4550>
```

In []: