Final Report

1.204

Serdar Colak

12/14/2011

PART 1 – Summary and Discussion

The paper entitled "The Price of Anarchy in Transportation Networks: Efficiency and Optimality Control" by Youn, Gastner and Jeong focuses on the effects of uncoordinated movement of individuals in transportation Networks.

It is obvious that among all options, humans choose the one that maximizes their utility. In the case of transportation Networks, this could be the path that minimizes cost, time of any combination of these. The issue here is that the superposition of all individuals making their own choices does not correspond to a cost minimization, or any other optimization for that matter. Transportation networks exhibit this phenomenon as well.
In the paper, networks that have delays associated with every link are investigated. In most transportation analysis, delay along a segment is accepted to be directly related to the flow along that segment. This makes obvious sense because as the number of cars on a highway increase, the average speed decreases, and consequently the delay increases. From a game theoretic perspective, an additional vehicle choosing the shortest path only makes everyone else suffer even higher delays. So the outcome is a Nash Equilibrium instead of a social optimum.

A simple example I went through is in Figure 1:

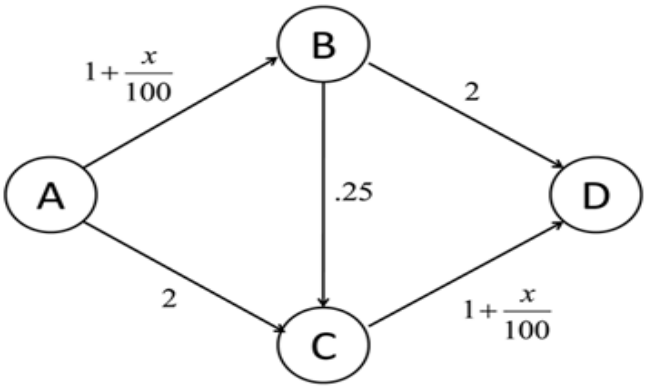


Figure 1. A sample network depicting flow of traffic along 4 nodes. The functions associated with every link represent the delays caused by the flow *x*

For this problem, if we are to assume that 100 drivers leave from A to go to D, then each will try to minimize their costs along their paths. There are three alternative paths: ABD, ACD and ABCD. The cost functions associated with each is given below:

$$C_{ABD} = 3 + \frac{x_{AB}}{100}, \qquad C_{ACD} = 3 + \frac{x_{CD}}{100}, \qquad C_{ABCD} = 2.25 + \frac{x_{AB} + x_{CD}}{100}$$

$$x_{AB} = f_{ABD} + f_{ABCD}, \qquad x_{CD} = f_{ACD} + f_{ABCD}.$$

$$C = \left(3 + \frac{x_{AB}}{100}\right) * f_{ABD} + \left(3 + \frac{x_{CD}}{100}\right) * f_{ACD} + \left(2.25 + \frac{x_{AB} + x_{CD}}{100}\right) * f_{ABCD}$$

Nash equilibrium occurs at the point where the flows are arranged in such a way that the costs associated with every path is equal. This is because if one path had suggested shorter delays, than that path would have already been chosen by the utility-maximizing individuals.

On the other hand, one can argue about a social optimum, by which he would mean minimizing the total delay experienced by all users, instead of minimizing for every individual. To do this for this problem, we would write the cost function as above, and then solve by imposing non-negativity constraint, and making sure that the sum of the flows are equal to 100 drivers.

	Social Optimum			**Nash Equilibrium**		
Path	Flow	Total Delay	Average Delay	Flow	Total Delay	Average Delay
ABD	50	350	3.50	25	375	3.75
ACD	50	350	3.50	25	375	3.75
ABCD	0	0	0	50	375	3.75

Table 1. Nash Equilibrium and Social Optimum outcomes for the network given in Figure 1. The average delay per person is higher for Nash Equilibrium.

An interesting point here is that the optimal solution totally disregards the link between nodes B and C. In other words, if the link hadn't been introduced, the drivers would be better off since the Nash Equilibrium would then be equal to the social optimum. This is referred to as the Braess' Paradox, which basically states that in fact, adding extra capacity to a network with selfish agents can sometimes reduce overall performance. The link between nodes B and C comply with this definition.

Finding the flows that correspond to a social optimum in given network can be generalized as below:

$$\min \sum_{link\ (i,j)} c_{ij}(f_{ij}) * f_{ij}$$

$$subject\ to \sum_{j \epsilon N(i)} f_{ij} = b_i, \quad for\ all\ nodes,$$
$$f_{ij} \geq 0, \quad for\ all\ links$$

For the Nash Equilibrium, one has to optimize what is called a projection function for every link, which basically is the integral of the associated cost function up to an arbitrary flow value.

$$\min \sum_{link\ (i,j)} \int_0^{f_{ij}} c_{ij}(f^*) * df^*$$

$$subject\ to \sum_{j \epsilon N(i)} f_{ij} = b_i, \quad for\ all\ nodes,$$
$$f_{ij} \geq 0, \quad for\ all\ links$$

At this point, authors make use of a measure named the Price of Anarchy, which measures the ratio of total costs in the two cases.

$$PoA = \frac{\sum c_{ij}\left(f_{ij}^{NE}\right) * f_{ij}^{NE}}{\sum c_{ij}\left(f_{ij}^{SO}\right) * f_{ij}^{SO}}$$

Using the ideas summarized above, the authors set out to investigate the simplified traffic networks of Boston, New York and London. The datasets are rigorously obtained by investigating the maps of each city in Google Earth, counting the number of lanes (which is multiplied by a constant of 2000 vehicles per lane to obtain the capacity,) measuring the length, identifying the direction of every link and assuming a speed limit of 35 mph. It is also assumed that the relationship between the flow and the delay along every link is given by

$$c_{ij}(f_{ij}) = \frac{d_{ij}}{v_{ij}}\left[1 + \alpha\left(\frac{f_{ij}}{p_{ij}}\right)^{\beta}\right]$$

The parameters are accepted as $\alpha = 0.2$ and $\beta = 10$, from previous works with this relationship. The findings are very interesting.

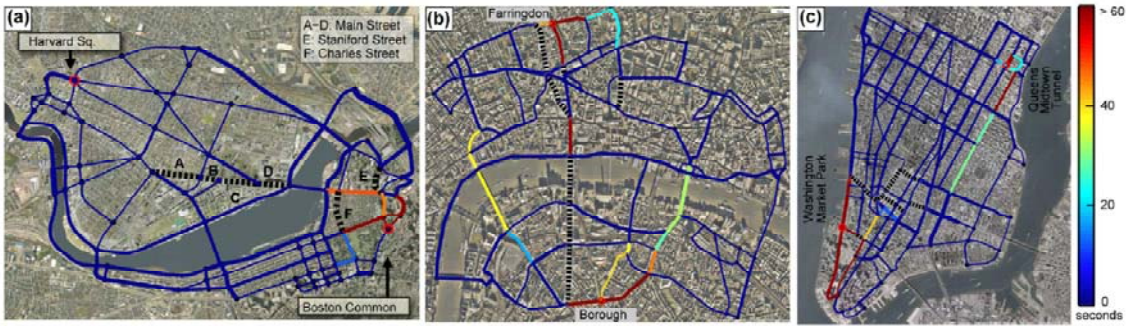


FIG. 2 (color online). Networks of principal roads (both solid and dotted lines; the thickness represents the number of lanes). (a) Boston-Cambridge area, (b) London, UK, and (c) New York City. The color of each link indicates the additional travel time needed in the Nash equilibrium if that link is cut [blue (dark gray): no change, red (medium gray): more than 60 seconds additional delay]. Black dotted lines denote links whose removal reduces the travel time, i.e., allowing drivers to use these streets in fact creates additional congestion. This counter-intuitive phenomenon is called "Braess's paradox."

It can be seen that certain links adhere to the Braess' paradox. For the case of Boston, the whole Main Street is in fact making the network less efficient, and pushing it away from the social optimum. The colors on every link suggest the amount of additional delay caused by the removal of that link. One critical point in this analysis is that it assumes a single source-sink pair. In other words, all drivers leave from Harvard Square and they all want to go to Boston Common. Obviously this is not the case in real life. In the traditional transportation analysis methods, all regions have a unique trip production value, and a trip attraction value. So applying multiple sources and sinks would definitely yield different results.

In terms of Price of Anarchy for increasing flows, all cities behave similarly. Boston reaches a global maximum PoA value at about 10000 vehicles. It is also noteworthy that there are a few local maxima with smaller flows. This response is an indication of how flows on links are redistributed as some of them become congested. Another interesting point is that the PoA approaches 1 for both very low and very high total flow values. This can be

explained by the fact that in the first case, no link exceeds capacity, and the delays caused by additional greedy drivers are so low that the difference between the Nash equilibrium and the social optimum is unnoticeably small. When the total flow is very high, the choices no longer make a huge difference because the main issue then is to be able to stay under capacity; and even the social optima lies very close to the capacity of the system.

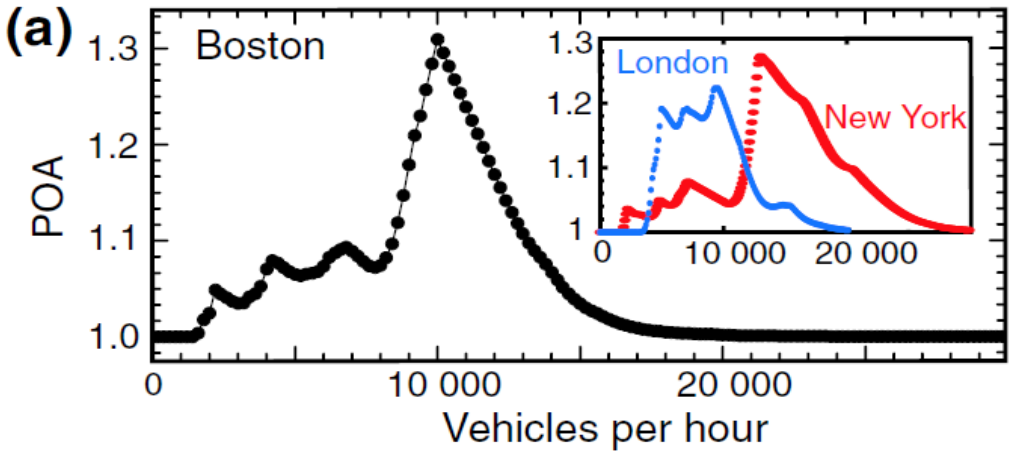


Figure 3. The PoA versus total flow relationship for the three cities. At its peak, Bostonians suffer 30% more delays because they act individually rather than a group. At very low or very high total flow values, this difference is nonexistent.

For the next step in the paper, authors carry the same type of analysis in undirected synthetic networks they create by simulations. Four types of networks: one dimensional lattices, Erdös-Renyi networks, Small worlds, and Barabasi-Albert models. Each of these networks has 100 nodes and 300 links. For this synthetic example, which I replicated later in this report, the delay function is assumed to be

$$c_{ij} = a_{ij} * f_{ij} + b_{ij}, \qquad where\ P(a_{ij}) = U[1,3], \qquad and\ \ P(b_{ij}) = U[1,100].$$

The constants are uniformly distributed discrete integers specific to every link. With this cost function, we can explicitly write our objective functions and the constraints for each of the two solutions. For Nash Equilibrium, it can be noted that the objective function is the integral of the linear cost curve from zero to f_{ij}. For the social optimum, the objective function is the total cost. The findings are depicted in Figure 4, where all four network types exhibit a similar response. It can be said, however, that small worlds have the highest PoA values, and the BA Models have the lowest. This finding is actually not very counterintuitive. In a small world, the long-range links immediately attract a lot of traffic because they are shortcuts. Consequently the demand for this link becomes so high that the delays along it increase proportionally. Therefore the network is moves away from the social optimum.

$$\boldsymbol{Nash\ Equilibrium}: \qquad \min \sum_{link\ (i,j)} \left(\frac{1}{2} a_{ij} f_{ij}^2 + b_{ij} f_{ij}\right)$$

$$subject\ to \sum_{j \epsilon N(i)} f_{ij} = b_i, \quad for\ all\ nodes,$$

$$f_{ij} \geq 0, \quad for\ all\ links.$$

$$\textbf{\textit{Social Optimum}}: \quad \min \sum_{link\ (i,j)} (a_{ij} f_{ij}^2 + b_{ij} f_{ij})$$

$$subject\ to \sum_{j \epsilon N(i)} f_{ij} = b_i, \quad for\ all\ nodes,$$

$$f_{ij} \geq 0, \quad for\ all\ links.$$

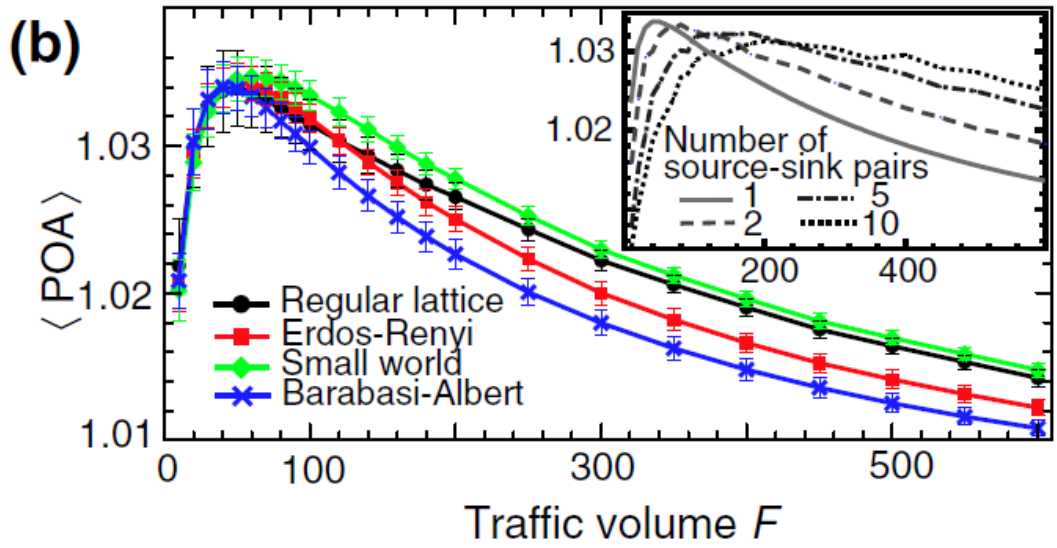


Figure 4. The PoA for different undirected network types with increasing total flow. Small world networks exhibit the highest PoA whereas BA models form the lower envelope.

My goals beforehand are to plot Figure 3 with the San Francisco data I have, and obtain the figure above. Typically, one should expect a PoA curve that starts and ends at zero, and contains several local maxima. The latter would, in itself, make sure that I can effectively create the four main network types with a given network size and a fixed number of links. Moreover, it would suggest me solving a convex cost network flow optimization problem on any network, and correctly adapting it to the case of undirected networks. Moreover, if I can, I will try to create some network instances in which I can visualize the Braess' Paradox.

PART 2 - Methodology and Experimentation

Replicating Figure 4

1. Creation of undirected lattices, small world, scale free and Erdös-Renyi networks with $n = 100$ and $k = 6$.
2. For each network, solving the optimization equation for both the Nash equilibrium and the social optimum to find the flow along every edge.
3. Calculating the total cost associated with each solution. Calculating the PoA.
4. Repeating parts 1-2-3 for many network instances to obtain averages.
5. Repeating parts 1-2-3-4 for many total flow values to plot the change of PoA with respect to each network

For the creation of scale-free and Erdös-Renyi networks, I used a function I wrote that uses a given degree sequence to create a single component random network without self loops and double edges. One can use the MATLAB random number generator to generate numbers from a Poisson distribution, which is a good approximation of the binomial distribution of a random network. For the scale free network, I first used the formula we discussed in class to write a function that returns the gamma value for a given network size and a desired average degree, and used that value to generate probabilities for every possible degree value. Once I obtained a random sample of 100 values with an average of 6, I created my scale free network.

For the small world networks, I used another function I wrote that uses the Watts-Strogatz model with a rewiring probability of 0.1, as suggested in the text. The lattices are small worlds with zero rewiring probability.

The optimization problems make up the biggest part of this project. The authors cite *Ahuja et al, 1993* for the implementation of minimum cost flow problems to networks with convex cost functions.

I found out that there are two algorithms that can be used to carry out this sort of analysis. One is called the successive shortest paths algorithm, and the other is the cycle cancelling algorithm. Both algorithms work with what is called a residual network, which is used to incorporate the unused capacity of every link as additional links in the opposite direction. I chose to implement the successive shortest paths algorithm, which requires to start with what's called a pseudoflow, then reduce costs as much as a shortest path length from one source to one sink. One continues this process iteratively until one reaches a point where the pseudoflow is a flow, that is, it satisfies the mass conservation principle.

The algorithm works for simple, directed networks. It is also successful at handling multiple sources, capacity constraints and different convex cost functions. The problem however is that it is difficult to adapt to undirected networks, because it suggests introducing new nodes to overcome the problem. This causes the network size to double, which makes the whole process longer. The text and the field does not really delve all that much into cases

with undirected networks and how they can be solved properly. After many hours, I chose to go along with the optimization toolbox in MATLAB, in which I introduced the mass conservation equations and the nonnegativity constraints.

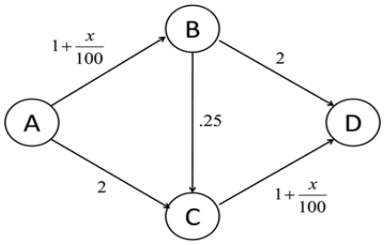
The optimization problem in matrix form becomes:

$$\min \sum_{link\ i} (\alpha_i f_i + \beta_i) * f_i$$

$$subject\ to\ I * f = b, \quad for\ all\ nodes,$$

$$f \geq 0, \qquad for\ all\ links$$

In this problem, f refers to the vector of link flows, I refers to an incidence matrix, and b refers to the demand at every node. An incidence matrix is an n by m matrix. Every column represents an edge, the element x referring to the origin of the link is 1, and the element y referring to the target of the link is -1. All other elements in the column is zero. One can immediately notice that the equation $I * f = b$ serves to fulfill the conservation of mass.



$$I = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

$$f_1 + f_2 = b_1$$

$$-f_1 + f_3 + f_4 = b_2$$

$$-f_2 - f_3 + f_4 = b_3$$

$$-f_3 - f_4 = b_4$$

The issue was to modify the network in a way so that its undirected nature could be captured. I altered the network by replacing every undirected edge with two directed edges in opposite directions, with identical cost functions. The network then had 2*m directed edges, and that many flow values to solve the optimization problem for. Once I obtained the values, I contracted the network back to its undirected state by finding the difference between every double link connecting the same node pair. Then I calculated the total cost, and consequently the PoA.

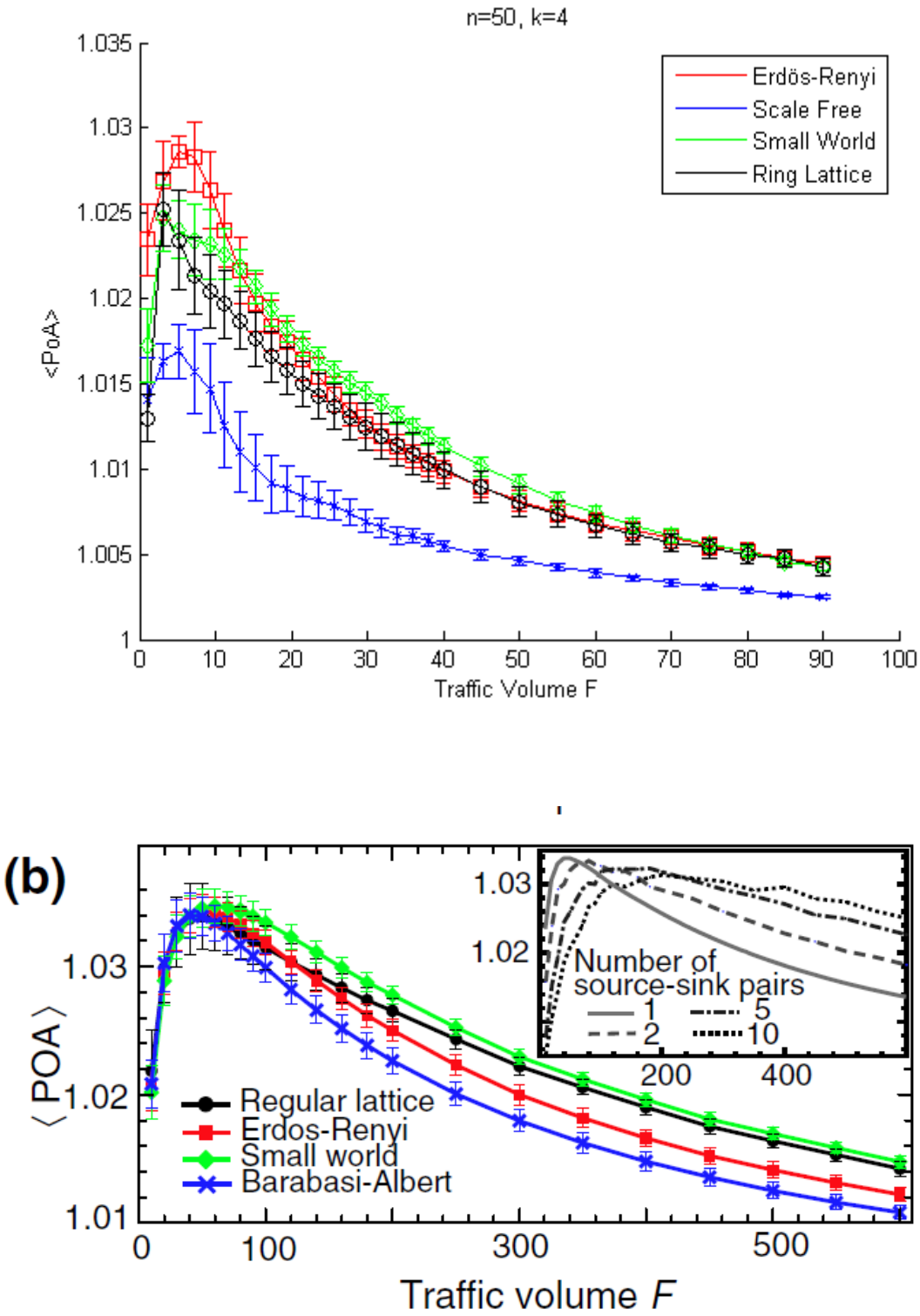


Figure 5. (a) My findings for how the average PoA changes with increasing traffic volume for different network types.

My findings are pretty consistent with the real, except a few factors. Due to time constraints, I had to cut back from the network size which might have altered with the result. I averaged over fewer node pairs, and fewer network instances. My guess is that with more iterations the curves will be smoother and approach those in the figure.

Reproducing Figure 3 for San Francisco

1. Integrating the cost function to get an explicit objective function for Nash equilibrium and social optimum.
2. Applying a minimum convex cost flow optimization
3. Finding average user costs for both solutions and find PoA for various flows.

$$C_{SO} = \sum \frac{d_{ij}}{v_{ij}} \left[1 + \alpha \left(\frac{f_{ij}}{p_{ij}} \right)^{\beta} \right] * f_{ij}$$

$$O_{NE} = \sum \frac{d_{ij}}{v_{ij}} \int_0^{f_{ij}} \left[1 + \alpha \left(\frac{f}{p_{ij}} \right)^{\beta} \right] df,$$

$$= \sum \frac{d_{ij}}{v_{ij} * (\beta + 1)} \left[f + \alpha f \left(\frac{f}{p_{ij}} \right)^{\beta} \right]_{f=0}^{f_{ij}}$$

$$= \frac{d_{ij}}{v_{ij} * (\beta + 1)} * f_{ij} * \left[1 + \alpha \left(\frac{f_{ij}}{p_{ij}} \right)^{\beta} \right]$$

Unfortunately my findings are incorrect for this objective function. Therefore I chose to use the linear delay function of the synthetic networks.

An important part of this process is the network simplification. To tackle the problem, I went through the network and removed links with low capacity. The point of doing this was to remove less important links and focus more on the highways, and also then I would have a smaller network.

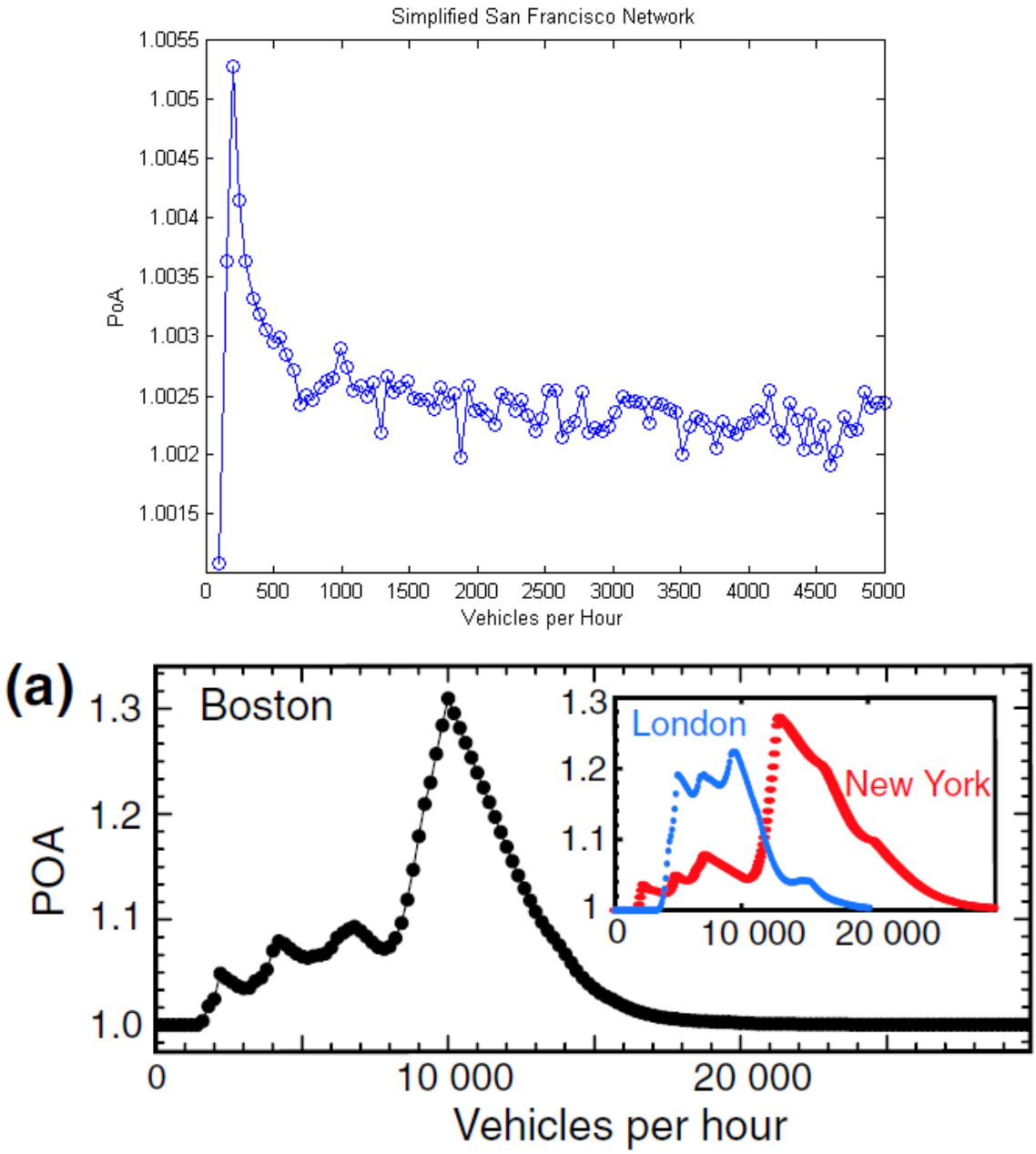


Figure 6. My findings for PoA vs. Vph for San Francisco, and the findings of the authors for Boston, London, New York.

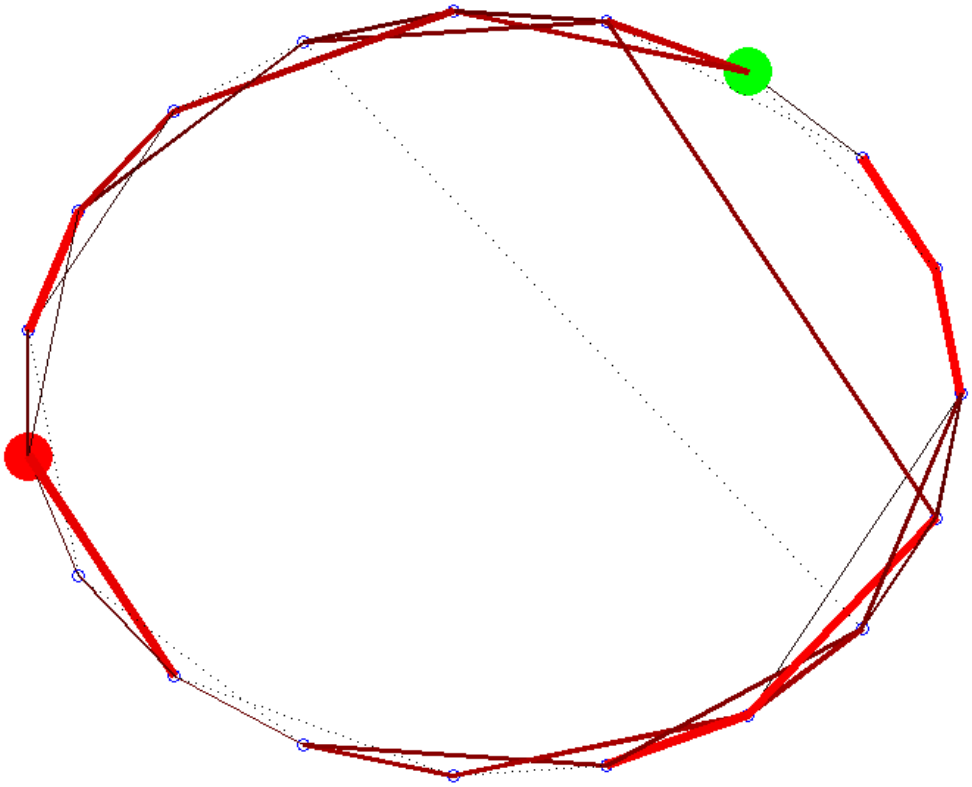
When compared, my results are obviously very different. This might be because of many reasons:

- My errors
- My choice to use a linear function
- The method of network aggregation
- My arbitrary selection of a start and endpoint

Searching for the Braess' Paradox in tiny small world networks

For this last exercise, I wanted to delve more into small world networks because the paper's findings suggested that these networks exhibited more PoA than any other. My argument was that considering that Braess' paradox is a concept that increases PoA, it should be more abundant in a small world network than the other network types.

In addition, small world networks are very easy to visualize, conceptualize, and comment on.



This figure is an example of what I wanted to visualize. The green node is the source, and the red node is the sink. The lightness of a link tells us how much additional delay its inexistence causes. Simply put, the lighter the link, the more delay people experience in its absence. The thickness of the link represents the magnitude of flow along it. As in the paper, dotted lines represent links that actually cause inefficiency. Note how a very central link has turned out to be a source of additional delays. While examining the network, it should also be noted that all links have distinct coefficients associated to their cost functions.

An interesting issue here is that the paradox is more difficult to encounter when the analysis is carried for many source-sink pairs. This makes sense, since then all links have different roles in each pair, and averaging over many also averages out the extreme effects of the existence of a link.

Conclusion

My goal in this project was to delve into the game-theoretic point of view the authors of this paper discuss about transportation networks. I was also hoping that this exercise would help me build tools which I can later use for network flow analysis and optimization.

Although I cannot say that I have a complete grasp of everything, I have managed to introduce myself to that area. I did a considerable amount of studying about several algorithms about convex cost flows, none of which I had the chance to implement into this report.

My findings are not impressive at all, but hopefully I will be able to build onto what I learned in this project and maybe obtain new findings that other people might try to replicate!

References

[1]Jeong et al., 2008, Price of Anarchy in Transportation Networks: Efficiency and Optimality Control

[2] Ahuja et al., 1993, Network Flows

Appendix- Matlab Codes:

Part 1:

```
%% This work is to compare PoA for all network types and reproduce Fig 3a.
% Type 1: Lattice
% Type 2: Erdos-Renyi.
% Type 3: Small World. (p=0.1)
% Type 4: Barabasi-Albert.
% All undirected
clear all
close all
n=50;
k=4;
for instance=1:5,
    for i=1:4
        m=0;
        if i==1,
            while m~=n*k
                adj=smallworld(n,k,0);
                m=sum(sum(adj));
            end
        elseif i==2,
            while m~=n*k
                adj=createnetwork(n,k,'poisson');
                m=sum(sum(adj));
            end
        elseif i==3,
            while m~=n*k
                adj=smallworld(n,k,0.1);
                m=sum(sum(adj));
            end
        elseif i==4,
            while m~=n*k
                adj=createnetwork(n,k2gamma(n,k),'scalefree');
                m=sum(sum(adj));
            end
        end
        m=sum(sum(adj));
        alfa=repmat(randi(3,[m/2,1]),2,1);
        beta=repmat(randi(10,[m/2,1]),2,1);
        pairs=combnk(randperm(n),2);
        pairs=pairs(unique(randi(length(pairs),[10,1])),:);
        pairno=size(pairs,1);
        maxsuprange=[linspace(1,40,20) linspace(45,90,10)];
        for maxsupindex=1:length(maxsuprange)
            maxsup=maxsuprange(maxsupindex);
            A=zeros(pairno,1);
            for pairind=1:pairno
                % Initialization
                I=adj2inc(adj);
                b=zeros(n,1);
                b(pairs(pairind,1))=maxsup;
                b(pairs(pairind,2))=-maxsup;
                options=optimset('Algorithm','interior-
point','Display','off');
                % Nash Equilibrium
                x=fmincon(@(f) sum(((alfa/2).*(f.^2))+(beta.*f))...
                    ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
                x=abs(x(1:end/2)-x((end/2+1):end));
                AUC_NE=sum((alfa(1:end/2).*x+beta(1:end/2)).*x)/maxsup;
```

```
                % Social Optimum
                x=fmincon(@(f) sum((alfa.*(f.^2))+(beta.*f))...
                    ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
                x=abs(x(1:end/2)-x((end/2+1):end));
                AUC_SO=sum((alfa(1:end/2).*x+beta(1:end/2)).*x)/maxsup;
                A(pairind)=AUC_NE/AUC_SO;
            end
            PoA(maxsupindex)=mean(A);
        end
        if i==1,
            RingLattice(:,instance)=PoA;
        elseif i==2,
            ErdosRenyi(:,instance)=PoA;
        elseif i==3,
            SmallWorld(:,instance)=PoA;
        elseif i==4,
            ScaleFree(:,instance)=PoA;
        end
    end
end
```

Part 2:

```
clear
load SanFranData.mat
sfnodes=find(inpolygon(BA_Node(:,3),BA_Node(:,2),sanfran(:,1),sanfran(:,2))
);
sfind=zeros(11305,1);
sfind(sfnodes)=1:length(sfnodes);
Adj=sparse(zeros(1189));
C=sparse(zeros(1189));
P=sparse(zeros(1189));
alfa=0.2;
beta=1;
for i=1:length(Attrs)
    s=Attrs(i,2);
    t=Attrs(i,3);
    stest=sum(sfnodes==s);
    ttest=sum(sfnodes==t);
    if stest==1 && ttest==1 && Attrs(i,8)>1900
        Adj(sfind(s),sfind(t))=1;
        C(sfind(s),sfind(t))=Attrs(i,4)/Attrs(i,5);
        P(sfind(s),sfind(t))=Attrs(i,8);
    end
end
%%
[S comp]=graphconncomp(Adj);
for i=1:965
    AAAA(i)=sum(comp==i);
end
Best=find(AAAA==max(AAAA));
rsfnodes=sfnodes(find(comp==Best));
rsfind=zeros(11305,1);
rsfind(rsfnodes)=1:length(rsfnodes);
Adj=sparse(zeros(151));
C=sparse(zeros(151));
P=sparse(zeros(151));
 for i=1:length(Attrs)
    s=Attrs(i,2);
    t=Attrs(i,3);
    stest=sum(rsfnodes==s);
```

```
    ttest=sum(rsfnodes==t);
    if stest==1 && ttest==1
        Adj(rsfind(s),rsfind(t))=1;
        C(rsfind(s),rsfind(t))=Attrs(i,4)/Attrs(i,5);
        P(rsfind(s),rsfind(t))=Attrs(i,8);
    end
 end
m=sum(sum(Adj));
I=adj2inc_dir(Adj);
c=60*nonzeros(C);
p=nonzeros(P);

%%
maxsuprange=linspace(100,5000,100);
b=zeros(151,1);
% alfa=randi(3,[m,1]);
% beta=randi(10,[m,1]);
for maxsupind=1:length(maxsuprange)
    maxsup=maxsuprange(maxsupind);
    b(84)=-maxsup;
    b(22)=maxsup;
    options=optimset('Algorithm','interior-point','Display','off');
    [x AUC_SO]=fmincon(@(f)
sum(c.*(1+alfa.*(f./p).^beta).*f),zeros(m,1),[],[],I,b,zeros(m,1),[],[],opt
ions);
    x=fmincon(@(f)
sum(c.*(f+(alfa.*(f.^(beta+1)))./((beta+1).*(p.^beta)))),zeros(m,1),[],[],I
,b,zeros(m,1),[],[],options);
    AUC_NE=sum(c.*(1+alfa.*(x./p).^beta).*x);
%                 x=fmincon(@(f) sum(((alfa/2).*(f.^2))+(beta.*f))...
%                     ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
%                 AUC_NE=sum((alfa.*x+beta).*x)/maxsup;
%                 % Social Optimum
%                 x=fmincon(@(f) sum((alfa.*(f.^2))+(beta.*f))...
%                     ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
%                 AUC_SO=sum((alfa.*x+beta).*x)/maxsup;
    PoA(maxsupind)=AUC_NE/AUC_SO;
    disp(PoA(maxsupind))
end
```

Part 3:

```
%% Braess' Paradox on a Small world network
clear all
n=20;
k=4;
m=0;
while m~=n*k
    Adj=smallworld(n,k,0.1);
    m=sum(sum(Adj));
end
theta=(0:(2*pi/(n-1)):(2*pi))';
coord=[cos(theta) sin(theta)];
Alfa=repmat(randi(3,[m/2,1]),2,1);
Beta=repmat(randi(10,[m/2,1]),2,1);
%% First: solve the original network to find the flow with the maximum PoA.
adj=Adj;
maxsuprange=0.1:0.1:4;
pairs=combnk(randperm(n),2);
pairs=pairs(unique(randi(length(pairs),[1,1])),:);
```

```
for maxsupindex=1:length(maxsuprange)
        maxsup=maxsuprange(maxsupindex);
        I=adj2inc(adj);
        b=zeros(n,1);
        alfa=Alfa; beta=Beta;
        b(pairs(1))=maxsup;
        b(pairs(2))=-maxsup;
        options=optimset('Algorithm','interior-point','Display','off');
        % Nash Equilibrium
        x=fmincon(@(f) sum(((alfa/2).*(f.^2))+(beta.*f))...
            ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
        x=abs(x(1:end/2)-x((end/2+1):end));
        AUC_NE=sum((alfa(1:end/2).*x+beta(1:end/2)).*x)/maxsup;
        % Social Optimum
        x=fmincon(@(f) sum((alfa.*(f.^2))+(beta.*f))...
            ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
        x=abs(x(1:end/2)-x((end/2+1):end));
        AUC_SO=sum((alfa(1:end/2).*x+beta(1:end/2)).*x)/maxsup;
        A=AUC_NE/AUC_SO;
    PoA(maxsupindex)=mean(A);
    disp(mean(A))
end
avgPoA=mean(PoA,2);
F=maxsuprange(find(avgPoA==max(avgPoA)));
%% Second: For the given F, find the additional delay caused by every link
% in the Nash equilibrium.
for mindex=m+1:-1:1
    adj=Adj;
    I=adj2inc(adj);
    I(:,mindex)=0;
    alfa=Alfa;
    beta=Beta;
    b=zeros(n,1);
    pairs=combnk(randperm(n),2);
    pairs=pairs(unique(randi(length(pairs),[1,1])),:);
    pairno=size(pairs,1);
    options=optimset('Algorithm','interior-point','Display','off');
    if mindex==m+1
        for pairind=1:pairno
            I=adj2inc(Adj);
            b=zeros(n,1);
            b(pairs(pairind,1))=F;
            b(pairs(pairind,2))=-F;
            x=fmincon(@(f) sum(((Alfa/2).*(f.^2))+(Beta.*f))...
            ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
            x=abs(x(1:end/2)-x((end/2+1):end));
            AUC_NE(pairind)=sum((Alfa(1:end/2).*x+Beta(1:end/2)).*x)/F;
        end
        Delay(mindex)=mean(AUC_NE);
        continue
    end

    for pairind=1:pairno
        b(pairs(pairind,1))=F;
        b(pairs(pairind,2))=-F;
        options=optimset('Algorithm','interior-point','Display','off');
        % Nash Equilibrium
        x=fmincon(@(f) sum(((alfa/2).*(f.^2))+(beta.*f))...
            ,zeros(m,1),[],[],I,b,zeros(m,1),[],[],options);
        x=abs(x(1:end/2)-x((end/2+1):end));
        AUC_NE(pairind)=sum((alfa(1:end/2).*x+beta(1:end/2)).*x)/F;
```

```
    end
    Delay(mindex)=mean(AUC_NE);
    disp(mean(AUC_NE))
end
%% Let's plot
NormDelay=Delay(1:end-1)-Delay(end);
Amat=zeros(n);
Amat(find(adj))=NormDelay;
Amat=Amat/max(max(Amat));
hold on
scatter(coord(:,1),coord(:,2))
linkwidthmax = 5;
widthmult = linkwidthmax/max(max(Amat));
[nodes nodet]=find(triu(Adj));
for link = 1:length(nodes)
    linkwidth = widthmult*Amat(nodes(link),nodet(link));
    if linkwidth>0
    plot([coord(nodes(link),1) coord(nodet(link),1)],...
        [coord(nodes(link),2) coord(nodet(link),2)],...
        '-','LineWidth',abs(linkwidth),'Color',[linkwidth/widthmult 0 0])
    axis([-1.5 1.5 -1.5 1.5])
    hold on
    elseif linkwidth<0
    plot([coord(nodes(link),1) coord(nodet(link),1)],...
        [coord(nodes(link),2) coord(nodet(link),2)],...
        'k--','LineWidth',2)
    hold on
    end
    if link==pairs(1),

plot(coord(link,1),coord(link,2),'go','MarkerSize',25,'MarkerFaceColor','g'
)
    elseif link==pairs(2),

plot(coord(link,1),coord(link,2),'ro','MarkerSize',25,'MarkerFaceColor','r'
)
    end
end
```