For office use only

T1 _____

T2 _____

T3 _____

T4 _____

Team Control Number

46648

Problem Chosen

D

For office use only

F1 _____

F2 _____

F3 _____

F4 _____

2016

ICM

Summary Sheet

The ways people spread information have changed a lot in the past centuries. From 1870s to 2010s, the media has developed from newspapers and radios to TV and the Internet. Our goal is to simulate the flow of information in different media throughout the history, and use our model to predict the information spreading in the future.

In our SIRQ model, we modified the original disease transmission SIR model to perform a continuous simulation of the information spreading process. In the spread of information, our $S(t)$ is defined as the number of people who do not know the information. $I(t)$ is the number of people who have received and is spreading the information. $R(t)$ is the number of people who have received and have spread but is no longer spreading the information. We defined an extra variable $Q(t)$, which is the number of all people who have received the information. We constructed differential equations to describe the change of the system. Solving the system allows us to obtain any macroscopic data of the system efficiently.

Although the SIRQ model is highly efficient, it does not provide any access to the microscopic value, such as the number of connections for each person. Besides, as information is spread discretely in the reality, the SIRQ model has additional error out of its continuity. To obtain full statistics on the system, we adopted the modified NW Small-World Network model to do the simulation. In our 1000-node directed network, each node represents a person and each edge represents a flow of information. Twenty groups with 50 nodes are formed. Different forms of networks are constructed for newspapers, radios, TVs and the Internet, and information is spread through the network in a different way for each media. We defined the value of information and the time delay for each spread. For newspapers, radios and TV, individuals cannot forward the information. In the Internet, we set up a set of rules to calculate the probability for a person to forward the information. Every source of information has a reliability, which depends on the form of message. Every person in the system is assigned a characteristic called opinion index with a beta distribution. Every time when a person receives a piece of information from a source, this person's opinion is changed by the product of the opinion index and the reliability of the source. When a person's opinion is changed by 100%, this person will believe this information. We set up three standards to judge the of flow of information: the number of people who receive the information, the number of people who believe the information, and the average time to receive the information. We use cross-validation to check the network model and the SIRQ model. The trends and results are similar in both models.

In question b and c, we are required to test our model's capability of prediction, and predict what will happen in 2050. We collected data from the Internet, of the percentage of households with each media. We used linear and non-linear regression to predict the future percentage of households. Then, with the predicted value, we computed the three standards values for the year 2014, the results are really close to the reality, with the largest error 4.6%. We defined the tightness of relationships as the total number of relationships in the network, and defined capacity as the maximum amount of information that can be spread in the network. Then, we adopted the regression model in question b to solve the problem.

In question d and e, we are required to study the factors affecting the opinions in the network. The results show that both of the value of information and the density of connections between people have positive effect on information influence.

Keywords: information spreading process, SIRQ model, NW Small-World Network, differential equations, linear regression, non-linear regression

Content

1 Problem Restatement

The ways people spread information have changed a lot in the past centuries. From 1870s to 2010s, the media has developed from newspapers and radios to TV and the Internet. Not only the speed of spreading information has been lifted, the range of spreading information has also been broadened. We focused on 1920s and on, because we only have access to data from then.

In this problem, we are asked to develop a model to study the flow of information and to determine which of the information can be called as news. Also, by studying past data, we are required to predict the change of flow of information as time goes, and verify our model with data of today's reality. Then, we are required to predict the social relationship and capacity of the flow of information in 2050. Finally, we are required to study the information influence on people of the information spreading process.

2 Model Design

2.1 SIRQ Model

As the spread of information has many similarities to the spread of a disease, we adopted the SIR epidemic model to simulate the information spreading process in the Internet. This provides us a macroscopic perspective. As is defined in the SIR epidemic model, $S(t)$ is the number of the susceptible, $I(t)$ is the number of infected people, and $R(t)$ is the number of people who were cured and has no effect on the system. To use this model, we assume that the number of people in the system is continuous.

In the spread of information, our $S(t)$ is defined as the number of people who do not know the information. $I(t)$ is the number of people who have received and is spreading the information. $R(t)$ is the number of people who have received and have spread but is no longer spreading the information. We defined an extra variable $Q(t)$, which is the number of all people who have received the information. $P(t)$ is the probability of a person to spread the information at time t. a is the average number of connections per person. N is the number of people in the system. T_d is the expected time delay for a person to see the information.

The probability $P(t)$ of a node to spread the information at time t is calculated as following:
We simulate the number of people who spread an information with negative exponential distribution since it decreases largely with time. The probability density function is $f(t) = me^{-mt}$. By looking at the data we have[3], we can see that 92.4% of the Internet users who forward a certain information do so in the first hour. We can then have $\int_0^{\frac{1}{24}} f(t)\,dt = 0.924$. By solving this equation, we have $m = 2.575$. Since the willingness of spreading the information is also related to the value k of the

information, the relationship between $P(t)$ and k can be written as $P(t) \propto ke^{-2.575t}$. Therefore, we determine $P(t) = cke^{-2.575t}$, where c is a constant.

In our differential equations,

$$\begin{cases} \dfrac{dS(t)}{dt} = -\dfrac{I(t)S(t)a}{NT_d} \\ \dfrac{dI(t)}{dt} = \dfrac{I(t)S(t)P(t)a}{NT_d} - \dfrac{I(t)}{T_d} \\ \dfrac{dR(t)}{dt} = \dfrac{I(t)}{T_d} \end{cases}$$

$Q(t)$ can be calculated as $Q(t) = N - S(t)$.

As the equation set has no exact solution, we use iterations to simulate the development of the variables.

2.2 Network Model

We use a modified NW Small-world Network to simulate social network.

2.2.1 Assumptions

a. Each person only receives information from two different types of sources in maximum, and he or she always chooses the later technologies.
   Justification: As a person always has more than one form of access to information, he or she will spend less time on the older technologies.
b. Every person who orders newspapers will read them.
   Justification: Newspaper is a daily-paid source of information. In common sense, people read the newspapers after they buy them.
c. The network always contains 20 groups with 50 people each.
   Justification: We use a small world with 1000 people to project the reality. We keep the size of the population constant so the change and effect of other variables can be studied.

2.2.2 Information Spreading Process

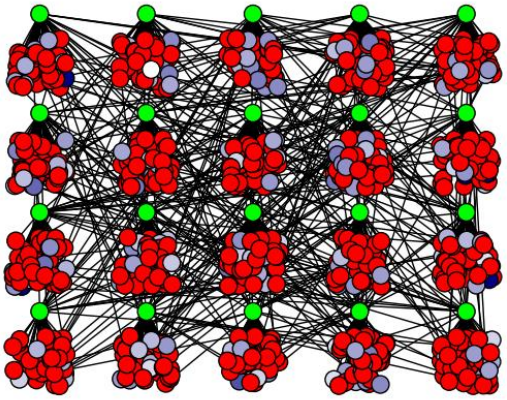
Small-world Construction:

We constructed a 1000-person directed network based on NW small-world network to represent social relationships. Each node represents a person. Each edge represents a flow of information. Twenty groups are formed, each group has 50 nodes.

- Newspapers:

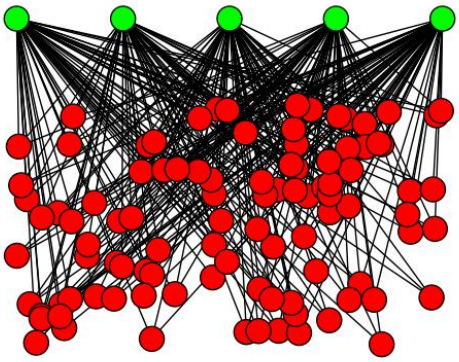
There is a newsstand in each "village". Most people (around 80 percent)[4] order local newspapers, and the others order newspapers from other "villages". Spreading newspapers between two nodes in the same village takes ΔT_{local} day(s), otherwise it takes ΔT_{far} day(s), $T_{local} < T_{far}$, ΔT_{local} and ΔT_{far} has a gamma distribution, $\Delta T_{local} \sim \Gamma(10T_{local}, 0.1)$, $\Delta T_{far} \sim \Gamma(10T_{far}, 0.1)$. Newspapers from different newsstands have different contents. The probability of a newspaper to contain a certain

piece of information is proportional to its value k, $k \in [0,1]$. Nodes representing people are not connected to each other. One person only connects to one newsstand[4].



- Radio/TV:

Every node connected to the same radio/TV channel receives the information in ΔT_r and ΔT_{tv} day(s) respectively, ΔT_r and ΔT_{tv} have gamma distributions, $\Delta T \sim \Gamma(10T_{r/tv}, 0.1)$. Different radio/TV channels have different contents. The probability of a radio/TV channel to contain a certain piece of information is proportional to its value k, $k \in [0,1]$. Nodes representing people are not connected to each other. One person connects to multiple radio channels. There is a probability P_r or P_{tv} for the audience to receive the information from one channel, because no one watches multiple channels at the same time all day long.



- The Internet:

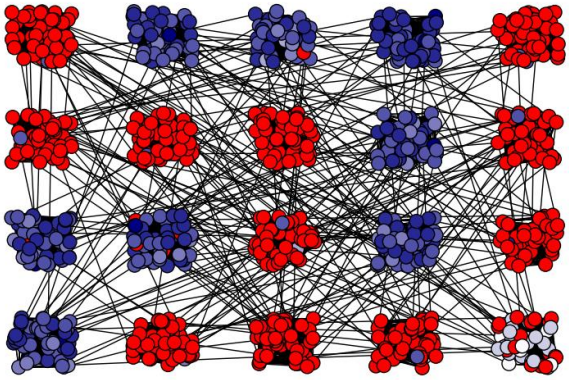
In the 1000-person small-world network, each node is connected to L_{local} nodes (L_{local} is an integer between 1 and 50) in the same "village". There are L_{far} edges generated randomly, connecting two nodes in different villages. Information can only be spread through an edge in a single direction. Information is spread in the Internet by the following steps:

a. Build an 1000-person small-world network, each node represents a person, and each edge represents the relationship between two people.
b. Color all nodes in red to represent they do not know the information.
c. Choose one node and color it in dark blue (blue represents the person has received the information). This node is the first spreader of the information. Set a timeline, and the current time $t = 0$ (counted in day(s)). Set a constant value $k \in [0,1]$ to represent the value of the information.

d. After a period of time Δt_i, the i_{th} node connected to a blue node sees the information. Δt has gamma distribution, $\Delta t \sim \Gamma(10T_d, 0.1)$, where T_d is the expected time delay. Color the i_{th} node in lighter blue after it sees the information, the darkness of blue is related to the time t_i on the timeline when node i sees the information.

e. Each blue node decides whether or not to spread this information. The probability $P(t)$ of a node to spread the information at time t is calculated by $P(t) = cke^{-2.575t}$, where c is a constant (derivation see 2.1.2).

f. If the node decides to spread the information, move back to step 4;
If the node does not decide to spread the information, this branch of information flow ends.



(L_{local}=20, L_{far}=250, k=0.5, c=0.3)

2.2.3 Filter

Since a piece of news can be easily spread to different regions, we use the number of "villages" which have received the information to determine which piece of information qualifies as news. When there are more than V villages have received the information, we define this information a news. From Figure 2.2.3, we can see that, when the value of a piece of information increases or when the density of connections between people increases, it is easier for the information to qualify as a piece of news.

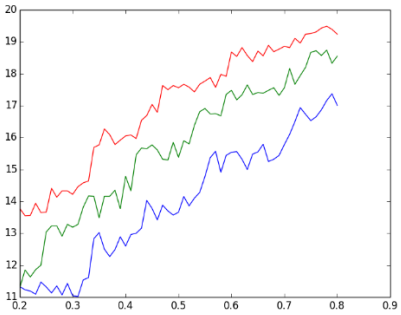


Figure 2.2.3

(x=k, y=number of "villages" received the information, T_d=0.08, c=0.3)

(blue: L_{local}=15, L_{far}=200; green: L_{local}=20, L_{far}=250; red: L_{local}=25, L_{far}=300)

2.2.4 Information Influence on People

Every source of information has a reliability r, it depends on the form of message. If the message is spread by newspapers, $r = r_n$, $r \in [0,1]$; if the message is spread by radio, $r = r_r$; If the message is spread by TV, $r = r_{tv}$; If the message is spread by the Internet, $r = r_i$.

Every person in the system is assigned a characteristic called opinion index, b, $b \in [0,1]$ and b has a beta distribution, $b \sim \beta(\alpha, \beta)$. It describes the initial bias of the person on this information, which is related to whether or not this person will believe this information.

Every time when a certain person with opinion index b_i receive a piece of information from a certain source with reliability r_j, it increases this person's opinion by $b_i \times r_j \times 100\%$.When a person's opinion is changed by 100%, this person will believe this information.

3 Model Testing

We define Q_{max} as the number of people who have received the information after the information stops spreading.

3.1 Mixed Forms of Information

From the data we got from the Internet[1][2][4], the percentages of households with newspapers, radios, TVs and the Internet by year are as following:

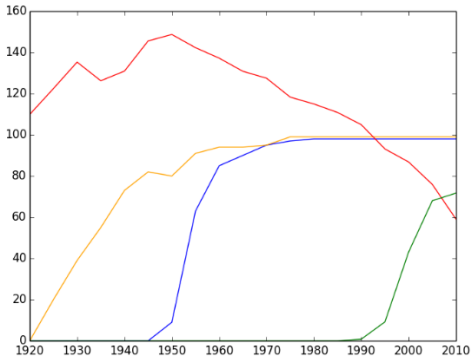


Figure 3.1

(x=year, y= percentage of households)

(red: newspapers; orange: radio; blue: TVs; green: the Internet)

3.2 Compare SIRQ Model and Network Model

In Figure 3.2.1 and figure 3.2.2, we compared our continuous model and the information spreading process in computer simulation. The dashed lines are the results of the SIRQ model, and the solid lines are the results of the spread of information in the network model (not a small-world). We studied the change of Q_{max} with time. Red lines represent the results when there are 30 connections per person; green lines represent the results when there are 40 connections per person; blue lines represent

the results when there are 50 connections per person. Figure 3.2.1 shows the change of Q_{max} with time when k=0.1. Figure 3.2.2 shows the change of Q_{max} with time when k=0.5. According to our results, the two models cross validated each other.

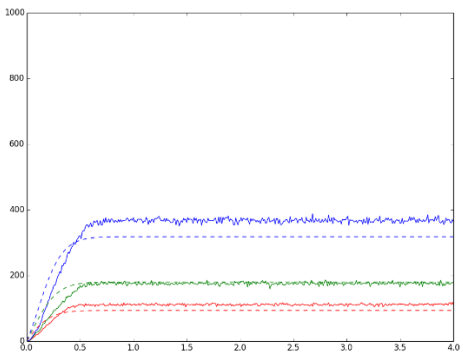


Figure 3.2.1

(x=time (day), y=Q_{max}, k=0.1, c=0.3)

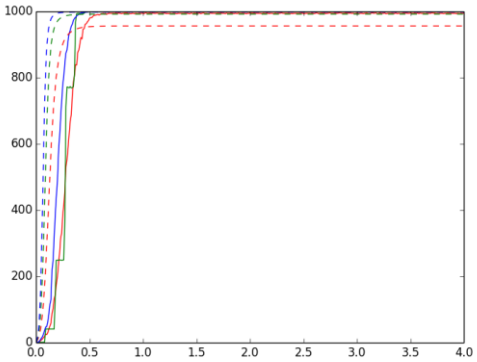


Figure 3.2.2

(x=time (day), y=Q_{max}, k=0.5, c=0.3)

3.3 Small-World Network

By testing the small-world network, we can see that both of the value k of an information and the density of connections between people have positive effect on Q_{max}.

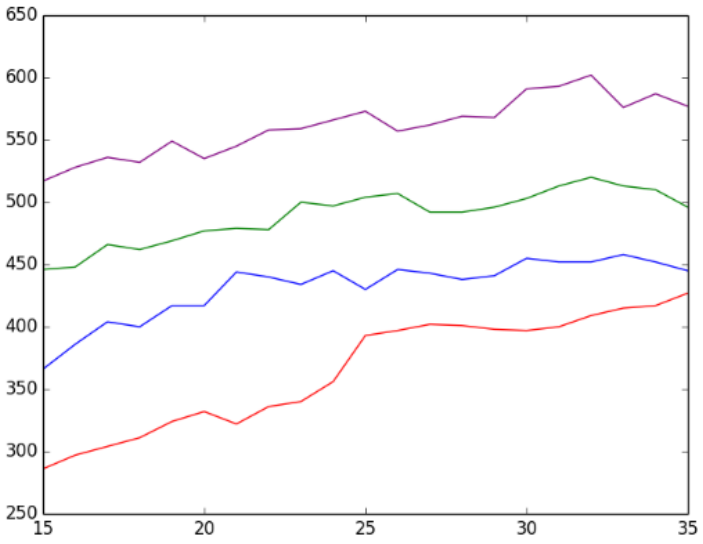


Figure 3.2

(x=L_{local}, y=Q_{max}, T_d=0.08, c=0.3)

(red: k=0.2; blue: k=0.4; green: k=0.6; purple: k=0.8)

3.4Historical Data

The following graphs take the data from 1920 to 2014(Figure 3.1) as input. We define B as the number of people who believe the information.

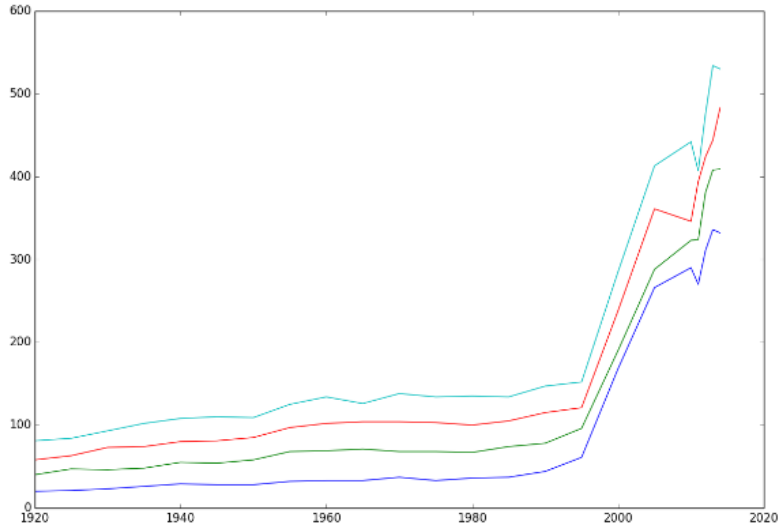


Figure 3.4.1

(y=B, c=0.3, T_d=0.08, L_{local}=25, L_{far}=300)

(dark blue: k=0.2; green: k=0.4; red: k=0.6; pale blue: k=0.8)

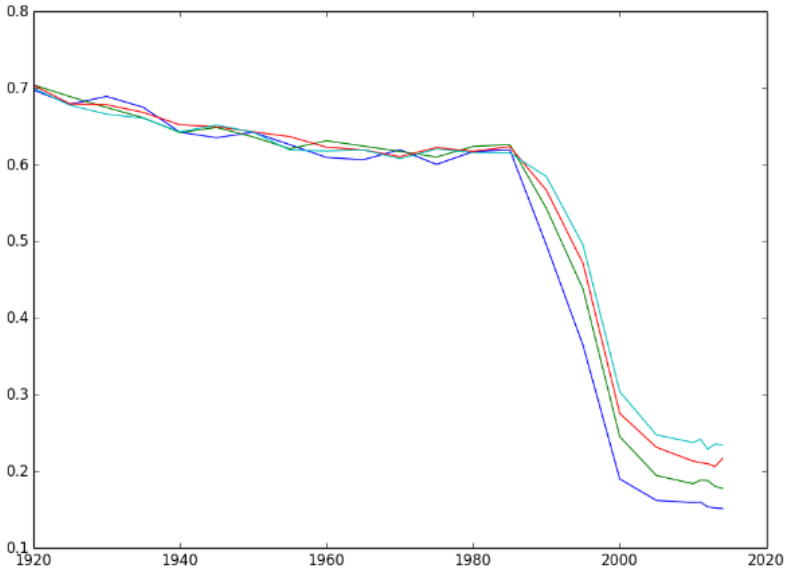


Figure 3.4.2

(y=ave. time to receive info, c=0.3, T_d=0.08, L_{local}=25, L_{far}=300)

(dark blue: k=0.2; green: k=0.4; red: k=0.6; pale blue: k=0.8)

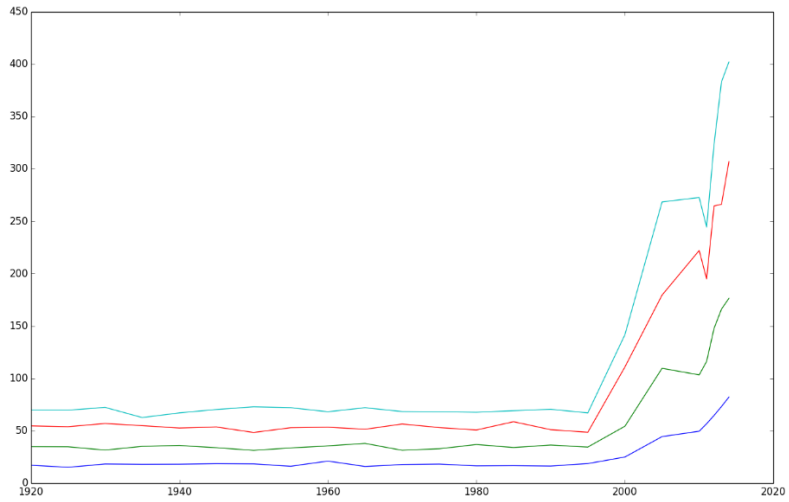


Figure 4.3.3

(y=Q_{max}, c=0.3, T_d=0.08, L_{local}=25, L_{far}=300)

(dark blue: k=0.2; green: k=0.4; red: k=0.6; pale blue: k=0.8)

The three graphs show the clear trend of spread of information with different values. At the point around 1995, when percentage of households of the Internet increases dramatically, each value has its most significant change.

4 Robustness Test

For our model, we'd like to analyze the robustness of our model. According to *Statistical Inference* by *George Casella* and *Roger L. Berger*, the robustness of a model is defined by the following:

a. It should have a reasonably good efficiency at the assumed model.

b. It should be robust in the sense that small deviations from the model assumptions should impair the performance only slightly.

c. Larger deviations from the model should not cause a catastrophe.

As a result, we examine our small-world network model with the three aspects of the definition:

a. In the assumed model (T_d=0.08, P_r=0.1, P_{tv}=0.04), we run the simulation 10 times with same condition settings. Then, we compare the new average time t, the number of people who have received the information after the information stops spreading Q_{max}, and the number of people who believe the information B to the original one we get every time.

	1	2	3	4	5	6	7	8	9	10
t	0.9725	0.9637	0.9589	0.9665	0.9677	0.9558	0.9800	0.9571	0.9801	0.9696
Q_{max}	605.84	609.22	608.66	619.71	614.18	603.88	606.05	608.46	620.69	618.38
B	124.95	124.86	126.99	126.40	126.36	123.37	126.22	127.77	126.48	128.55

b. With some small deviations of time delay to receive the information T_d, probability P_r or P_{tv} for the audience to receive the information from one channel, we compare the new average time t, the number of people who have received the information after the information stops spreading Q_{max}, and the number of people who believe the information B to the original one.

	$T_d + 5\%$	$T_d - 5\%$	$P_r + 5\%$	$P_r - 5\%$	$P_{tv} + 5\%$	$P_{tv} - 5\%$	**Original**
t	0.9521	0.9606	0.9627	0.9643	0.9625	0.9740	**0.9769**
Q_{max}	602.80	605.62	607.38	691.25	610.08	610.45	**605.91**
B	119.08	122.98	126.96	128.72	127.84	127.07	**125.51**

c. With some large deviations of time delay to receive the information T_d, probability P_r or P_{tv} for the audience to receive the information from one channel, we compare the new average time t, the number of people who have received the information after the information stops spreading Q_{max}, and the number of people who believe the information B to the original one.

	$T_d + 50\%$	$T_d - 50\%$	$P_r + 50\%$	$P_r - 50\%$	$P_{tv} + 50\%$	$P_{tv} - 50\%$	**Original**
t	0.9816	0.9387	0.9790	0.9619	0.9569	0.9475	**0.9769**
Q_{max}	588.90	658.55	616.25	603.61	637.86	571.28	**605.91**
B	108.31	160.88	130.38	121.84	130.21	116.90	**125.51**

We can see from the table that the model is robust.

5 Study of the Trends in 1920-2014 (Question b)

To validate our model, we predicted the information spreading situation in 2014, and checked the results the data we acquired[4]. To obtain a fair prediction, we predicted the percentages of households with different types of sources.

For TVs and radios, the ratio stayed constant for the last decades, so we assumed that the ratio will also be constant in the future. For newspapers, as the percentage has decreased since 1950, we use linear fit function in Mathematica to predict its percentage in 2016 (Figure 5.1). The linear function is $2978.4369 - 1.4484 \times i$. According to this linear function, 60.9% of the families are predicted to be reading newspapers in 2014.

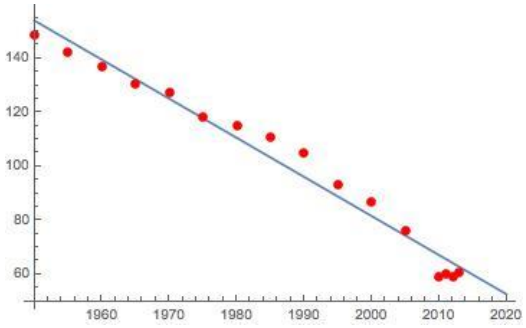


Figure 5.1

The Internet is currently in the burgeoning stage. Figure 3.1 shows that the burgeoning stages of TVs, radios and the Internet are very similar. Therefore, we predicted that the future development of the Internet by fitting the development pattern of TVs into the Internet. The increase in the percentage of households with TVs is fitted with the function $a - b \times e^{-cx+d}$ (Figure 5.2). By using Mathematica, a=99.8337, b=66.6891, c=0.0996124, d=199.026. Then, the graph is transitioned in the coordinates to the best fit curve to fit the percentage of the Internet (Figure 5.3). In 2014, 86.2856% users are predicted to have access to Internet according to our prediction.

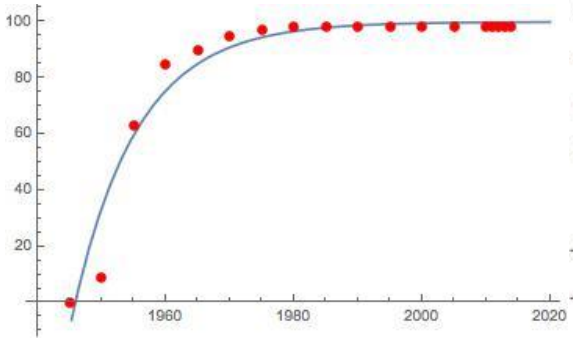


Figure 5.2

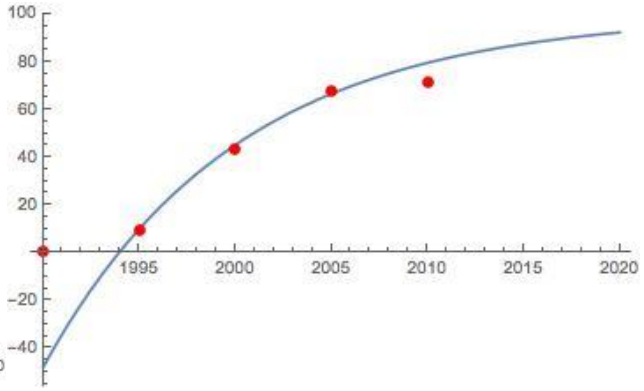


Figure 5.3

With the predicted percentages of households with different types of sources, we run our simulation for 100 times, and take the average to obtain the three standards to judge the spread of information. (We define B as the number of people who believe the information)

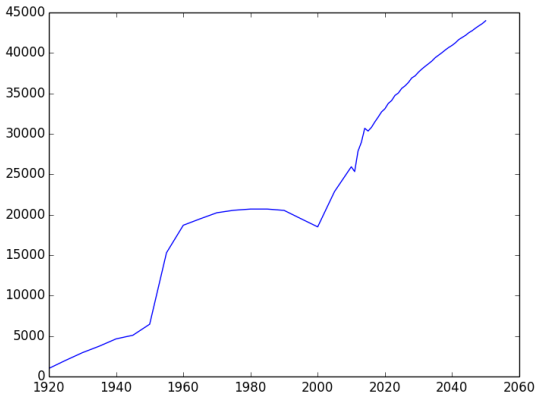
	B	Q_{max}	ave. time to receive info
Predicted	156	439	0.1976
Real	162	450	0.1889
Error	3.7%	2.4%	4.6%

Since the value is really close to the real value, our model has good prediction capability.

6 Situation in 2050 (Question c)

6.1 Relationship

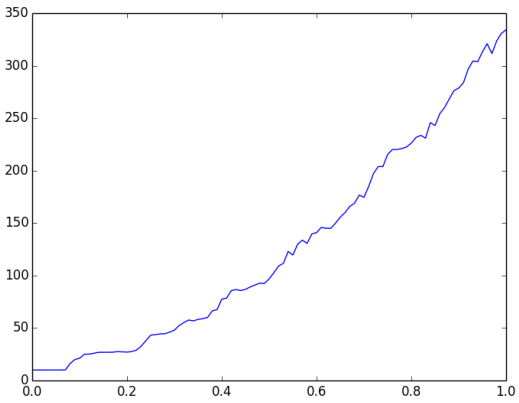
To predict what happens in 2050, we used our regression method in question b to predict the percentages of households with the Internet and newspapers. To measure the relationship among people in the network, we describe the tightness of the relationships by the total number of edges in the 1000-person small-world network.



According to the simulation, the tightness of relationships in 2050 is almost twice as much as we have now. As we controlled all the other parameters constant, the results can be considered a conservative estimation of the tightness of the relationships.

6.2 Capacity

From the data we have, on average, an Internet user forwards 1.7 pieces of information per day[3]. We put information with different values into the network, and let the information spread. As each information stops spreading, a number of users forward the information, causing the average forwarding number to increase. When the average forwarding number increases to 1.7, the network is saturated with its full capacity. By counting the number of forwarding number for each given information value between 0 and 1, we plot a graph of the data.



Such set of data can help us determine the maximum number of information that can be spread through the network. For example, when the value of the information is 0.2, the corresponding number of forwarding is 27.09. In a network of 1000 people, the total number of forwarding is 1700. Roughly, the maximum number of information with value 0.2 that can be spread in the network is 63.

7 Information Influence (Question d)

We define B as the number of people who believe the information.

7.1 Value of the Information

In 2014, if each radio audience listens to 5 radio channels and each TV audience watches 20 TV channels, the relationship between the number of people who believe the information and the value of this information is shown in Figure 7.1.

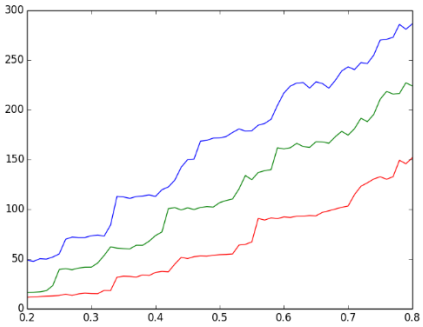


Figure 7.1

(x=k, y=B, c=0.3, $year$=2014, T_d=0.08, P_r=0.1, P_{tv}=0.04)

(red: L_{local}=15, L_{far}=200; green: L_{local}=20, L_{far}=250; blue: L_{local}=25, L_{far}=300)

7.2 Density of Connections Between People

In 2014, if each radio audience listens to 5 radio channels and each TV audience watches 20 TV channels, the relationship between the number of people who believe the information and the density of connections between people is shown in Figure 7.2.

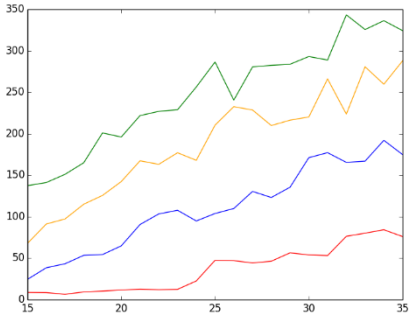


Figure 7.2

(x=L_{local}, y=B, c=0.3, $year$=2014, T_d=0.08, P_r=0.1, P_{tv}=0.04)

(red: k=0.2; blue: k=0.4; orange: k=0.6; green: k=0.8)

8 Determinants (Question e)

8.1 Value of Information

The dashed lines show the number of people who have received the information after the information stops spreading Q_{max}, and the solid lines show the number of people who believe the information B. We studied the change of Q_{max} and B for information with different value k. Red lines represent the results when there are 15 local connections per person and 200 far connections in the system; green lines represent the results when there are 20 local connections per person and 250 far connections in the system; blue lines represent the results when there are 25 local connections per person and 300 far connections in the system. According to our results, both k and L have positive effect on Q_{max} and B.

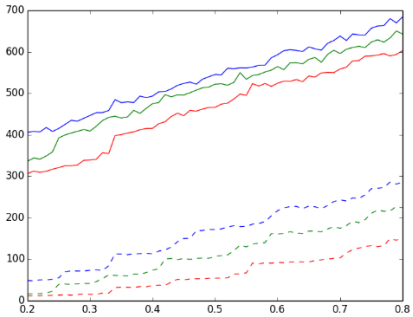


Figure 8.1

(x=k, y_{solid}=Q_{max}, y_{dash}=B, c=0.3, $year$=2014, T_d=0.08, P_r=0.1, P_{tv}=0.04)

(red: L_{local}=15, L_{far}=200; green: L_{local}=20, L_{far}=250; blue: L_{local}=25, L_{far}=300)

8.2 Initial Opinion

Beta distribution is a distribution for variable between 0 and 1. There are two parameters for the distribution: α and β, while $\frac{\alpha}{\alpha+\beta}$ is the expected value of the distribution. Two control the expected value and keep the variance stable, we keep $\alpha + \beta = 1$, so alpha is always the expected value of the distribution.

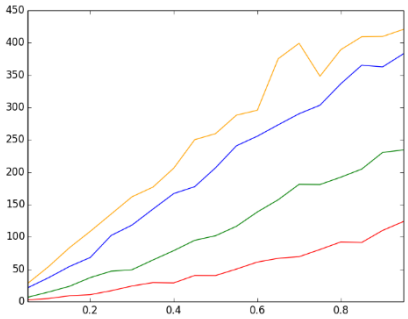


Figure 8.2

(x=α, y=B, c=0.3, $year$=2014, T_d=0.08, P_r=0.1, P_{tv}=0.04)

(red: k=0.2; green: k=0.4; blue: k=0.6; orange: k=0.8)

9 Strengths and Weaknesses

9.1 SIRQ Model

9.1.1 Strengths

a. The model has immediate simulation and calculation of a huge information network.
b. The model is highly stable, and it is not sensitive to changes in small ranges.
c. The model embodies greatly the relationships among macroscopic values (e.g. the number of people who have received the information).
d. The model shows a clear trend of the variables after solved.

9.1.2 Weaknesses

a. The model does not consider specific relationships when a piece of information is transferred between people.
b. The model does not provide access to any specific information in the system for a certain person.
c. The model cannot be visualized.

9.2 Network Model

9.2.1 Strengths

a. The simulation is highly similar to the reality. Every iteration means a time period in the reality.
b. Each node updates its property in each iteration. Therefore, any information in any iteration of any node can be obtained.
c. The model is highly visible. With proper tools, the networks, relationships and spreading process be visualized.
d. The simulation allows events to happen according to a probability. Such probability also adds reasonable noise to the input.

9.2.2 Weaknesses

a. Comparatively inefficient, as the whole situation needs to be simulated again for an iteration. Also, to stabilize the output, the experiment is repeated and taken average, so the efficiency is lower in this case.
b. The trends cannot be easily obtained before finishing the simulation.

References

[1] Worldbank.org, (2016). *Internet users (per 100 people)*. [online] Available at: http://data.worldbank.org/indicator/IT.NET.USER.P2/countries/1W?page=3&display=default

[2] Pew Research Center, (2016). *Newspapers: Circulation by Publication Type.* [online] Available at: http://www.journalism.org/media-indicators/newspaper-circulation/

[3] Stadd, A. (2013). *50 Twitter Fun Facts.* [online] Social Times. Available at: http://www.adweek.com/socialtimes/50-twitter-fun-facts/475073

[4] U.S. Census Bureau. (1999). *20th Century Statistics*. [online] Available at: https://www.census.gov/prod/99pubs/99statab/sec31.pdf

Appendix A (data)

Explosiveness: 0.2/0.4/0.6/0.8
Data Type: Spread Count
[286, 297, 304, 311, 324, 332, 322, 336, 340, 356, 393, 397, 402, 401, 398, 397, 400, 409, 415, 417, 427]
[366, 386, 404, 400, 417, 417, 444, 440, 434, 445, 430, 446, 443, 438, 441, 455, 452, 452, 458, 452, 445]
[446, 448, 466, 462, 469, 477, 479, 478, 500, 497, 504, 507, 492, 492, 496, 503, 513, 520, 513, 510, 496]
[517, 528, 536, 532, 549, 535, 545, 558, 559, 566, 573, 557, 562, 569, 568, 591, 593, 602, 576, 587, 577]
Link:
[15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]

Link: 15/20/25
Data Type: Trust Count
[11.33, 11.23, 11.19, 11.09, 11.47, 11.32, 11.13, 11.35, 11.07, 11.43, 11.04, 11.02, 11.54, 11.61, 12.83, 13.02, 12.5, 12.27, 12.49, 12.89, 12.6, 12.96, 13.0, 13.16, 14.03, 13.78, 13.42, 13.88, 13.7, 13.57, 13.66, 14.15, 13.86, 14.1, 14.28, 14.78, 15.37, 15.57, 14.91, 15.44, 15.54, 15.56, 15.32, 15.0, 15.48, 15.55, 15.79, 15.25, 15.32, 15.44, 15.78, 16.1, 16.5, 16.94, 16.73, 16.53, 16.65, 16.87, 17.16, 17.37, 17.01]
[11.28, 11.85, 11.63, 11.86, 12.0, 13.04, 13.23, 13.23, 12.91, 13.28, 13.19, 13.28, 13.82, 14.17, 14.16, 13.49, 14.16, 14.16, 14.35, 13.77, 14.78, 14.33, 15.46, 15.67, 15.65, 15.77, 15.61, 15.32, 15.3, 15.85, 15.38, 15.9, 15.8, 16.38, 16.81, 16.91, 16.74, 16.75, 16.68, 17.35, 17.48, 17.18, 17.34, 17.65, 17.35, 17.41, 17.39, 17.48, 17.56, 17.32, 17.56, 18.16, 17.67, 17.94, 18.2, 18.67, 18.72, 18.57, 18.74, 18.33, 18.55]
[13.78, 13.55, 13.56, 13.94, 13.65, 13.66, 14.41, 14.13, 14.33, 14.33, 14.22, 14.46, 14.58, 14.64, 15.69, 15.77, 16.27, 16.09, 15.78, 15.92, 16.05, 16.08, 15.97, 16.54, 16.69, 17.04, 16.79, 17.63, 17.5, 17.63, 17.56, 17.67, 17.58, 17.43, 17.67, 17.77, 17.88, 17.57, 17.98, 17.92, 18.68, 18.54, 18.82, 18.57, 18.38, 18.71, 18.56, 18.89, 18.69, 18.77, 18.86, 18.82, 19.11, 18.96, 19.24, 19.26, 19.31, 19.43, 19.49, 19.39, 19.24]
explosiveness:
[0.2, 0.2100, 0.2200, 0.2300, 0.2400, 0.2500, 0.2600, 0.2700, 0.2800, 0.2900, 0.3000, 0.3100, 0.3200, 0.3300, 0.3400, 0.3500, 0.3600, 0.3700, 0.3800, 0.3900, 0.4000, 0.4100, 0.4200, 0.4300, 0.4400, 0.4500, 0.4600, 0.4700, 0.4800, 0.4900, 0.5000, 0.5100, 0.5200, 0.5300, 0.5400, 0.5500, 0.5600, 0.5700, 0.5800, 0.5900, 0.6000, 0.6100, 0.6200, 0.6300, 0.6400, 0.6500, 0.6600, 0.6700, 0.6800, 0.6900, 0.7000, 0.7100, 0.7200, 0.7300, 0.7400, 0.7500, 0.7600, 0.7700, 0.7800, 0.7900, 0.8]

Explosiveness: 0.2/0.4/0.6/0.8
Data Type: Spread Count
[20, 21, 23, 26, 29, 28, 28, 32, 33, 33, 37, 33, 36, 37, 44, 61, 168, 266, 290, 270, 311, 336, 332]
[40, 47, 46, 48, 55, 54, 58, 68, 69, 71, 68, 68, 67, 74, 78, 96, 190, 288, 323, 324, 381, 408, 409]
[58, 63, 73, 74, 80, 81, 85, 97, 102, 104, 104, 103, 100, 105, 115, 121, 238, 361, 346, 394, 424, 444, 483]
[81, 84, 93, 102, 108, 110, 109, 125, 134, 126, 138, 134, 135, 134, 147, 152, 285, 413, 442, 407, 476, 534, 530]
Longterm Timeline:
[1920, 1925, 1930, 1935, 1940, 1945, 1950, 1955, 1960, 1965, 1970, 1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2011, 2012, 2013, 2014]

Explosiveness: 0.2/0.4/0.6/0.8
Data Type: Average Spread Time
[0.6976, 0.6790, 0.6891, 0.6749, 0.6424, 0.6352, 0.6423, 0.6261, 0.6094, 0.6064, 0.6195, 0.6003, 0.6170, 0.6194, 0.4948, 0.3640, 0.1900, 0.1619, 0.1591, 0.1598, 0.1536, 0.1520, 0.1514
[0.7043, 0.6889, 0.6746, 0.6613, 0.6419, 0.6486, 0.6362, 0.6207, 0.6312, 0.6243, 0.6173, 0.6099, 0.6238, 0.6262, 0.5427, 0.4375, 0.2452, 0.1944, 0.1837, 0.1883, 0.1878, 0.1803, 0.1776
[0.7040, 0.6793, 0.6783, 0.6682, 0.6520, 0.6494, 0.6429, 0.6366, 0.6228, 0.6193, 0.6104, 0.6223, 0.6174, 0.6235, 0.5665, 0.4710, 0.2755, 0.2312, 0.2133, 0.2113, 0.2097, 0.2062, 0.2166
[0.7002, 0.6775, 0.6659, 0.6606, 0.6429, 0.6516, 0.6425, 0.6189, 0.6178, 0.6193, 0.6078, 0.6204, 0.6158, 0.6152, 0.5843, 0.4950, 0.3038, 0.2474, 0.2375, 0.2417, 0.2288, 0.2356, 0.2345
Longterm Timeline:
[1920, 1925, 1930, 1935, 1940, 1945, 1950, 1955, 1960, 1965, 1970, 1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2011, 2012, 2013, 2014]

Explosiveness: 0.2/0.4/0.6/0.8
Data Type: Trust Count
[137.5, 141.2, 151.1, 165.1, 201.0, 196.1, 221.9, 227.0, 228.9, 256.4, 286.5, 240.6, 280.6, 282.5, 283.8, 293.1, 288.9, 343.3, 325.6, 336.3, 324.3]

[67.9, 91.2, 97.3, 115.1, 125.7, 142.4, 167.4, 163.2, 177.1, 168.0, 210.4, 232.7, 228.6, 209.9, 216.4, 220.2, 266.1, 223.7, 280.7, 259.7, 287.9]
[24.5, 38.5, 43.3, 53.5, 54.4, 64.9, 90.6, 103.4, 107.8, 94.9, 103.9, 109.9, 130.5, 123.2, 135.6, 171.2, 177.2, 165.5, 167.0, 192.1, 174.9]
[8.7, 8.5, 6.5, 9.3, 10.4, 11.7, 12.7, 12.1, 12.5, 22.6, 47.5, 47.1, 44.3, 46.4, 56.5, 54.0, 53.2, 76.4, 80.2, 84.3, 76.1]
Link:
[15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]

Link: 15/20/25
Data Type: Spread Count
[306.4200, 312.37, 309.2999, 311.9700, 316.9500, 320.8600, 325.19, 325.38, 327.31, 338.37, 338.9600, 340.8299, 356.8199, 354.7000, 398.1600, 400.2599, 404.2200, 406.8600, 412.0199, 415.19, 415.3400, 426.63, 431.3500, 444.0299, 452.0099, 445.9499, 458.6800, 456.9700, 462.0, 465.7799, 465.9900, 473.8000, 476.2400, 486.0099, 498.0699, 495.12, 523.3199, 517.25, 523.6699, 516.5199, 524.2999, 529.0499, 528.8199, 532.9700, 527.9300, 541.7000, 539.13, 549.0799, 550.2599, 549.7999, 558.5, 562.6699, 578.2000, 578.6799, 589.4500, 589.8099, 592.1500, 595.5099, 590.0, 593.5999, 602.7899
[336.3999, 343.9300, 341.25, 348.8999, 358.75, 392.3299, 399.5400, 404.2200, 408.3199, 412.6499, 408.9299, 420.5799, 434.1999, 442.0900, 444.6900, 440.2599, 443.1100, 458.87, 451.4900, 463.8400, 474.7599, 477.81, 496.7300, 491.3500, 496.3600, 495.9499, 501.3999, 507.56, 513.1900, 515.0499, 521.8399, 522.9400, 519.3999, 526.2100, 549.7100, 533.7200, 542.87, 545.2899, 551.3099, 555.75, 564.4800, 556.8299, 573.5799, 573.88, 570.8899, 581.75, 586.2100, 574.63, 593.5799, 603.7699, 595.6800, 606.2300, 610.2899, 613.25, 610.0799, 623.4400, 628.8400, 623.2100, 633.4800, 650.25, 642.7799
[405.9099, 407.9099, 406.9900, 417.5800, 407.8999, 415.2900, 424.8500, 434.7999, 432.8400, 439.63, 446.5099, 453.3400, 453.38, 458.6399, 484.6399, 477.3899, 479.1800, 477.6999, 493.2400, 489.3999, 493.2599, 503.5099, 504.1399, 510.5299, 519.12, 523.8999, 526.9400, 521.9400, 534.0, 539.7699, 545.3500, 544.3600, 560.3500, 559.2400, 561.1800, 560.8800, 563.25, 567.2799, 567.8900, 585.6100, 592.8900, 602.7200, 604.9300, 603.3400, 600.7999, 611.7300, 607.1900, 603.8899, 620.4900, 627.2000, 638.0900, 627.2400, 642.7899, 640.88, 640.38, 656.8500, 662.0900, 663.0299, 679.9600, 669.6600, 684.0399
Explosiveness:
[0.2, 0.2100, 0.2200, 0.2300, 0.2400, 0.2500, 0.2600, 0.2700, 0.2800, 0.2900, 0.3000, 0.3100, 0.3200, 0.3300, 0.3400, 0.3500, 0.3600, 0.3700, 0.3800, 0.3900, 0.4000, 0.4100, 0.4200, 0.4300, 0.4400, 0.4500, 0.4600, 0.4700, 0.4800, 0.4900, 0.5000, 0.5100, 0.5200, 0.5300, 0.5400, 0.5500, 0.5600, 0.5700, 0.5800, 0.5900, 0.6000, 0.6100, 0.6200, 0.6300, 0.6400, 0.6500, 0.6600, 0.6700, 0.6800, 0.6900, 0.7000, 0.7100, 0.7200, 0.7300, 0.7400, 0.7500, 0.7600, 0.7700, 0.7800, 0.7900, 0.8]

Link: 15/20/25
Data Type: Trust Count
[11.4600, 11.84, 12.24, 12.6200, 12.8700, 13.3099, 14.6400, 13.34, 14.92, 15.66, 15.23, 15.1300, 18.3699, 18.2599, 31.6400, 32.6899, 32.4600, 31.7899, 33.9899, 33.3999, 36.4699, 37.4600, 37.0400, 44.8100, 51.6200, 50.3800, 52.32, 52.9800, 52.7699, 53.82, 54.3299, 54.5300, 55.0099, 63.9500, 64.6700, 67.1400, 90.7399, 89.1400, 91.2899, 90.5800, 92.2399, 91.7199, 93.0, 93.2199, 93.6599, 93.4099, 96.8700, 98.4099, 100.1800, 101.97, 103.2100, 114.77, 123.12, 126.55, 130.4000, 132.6499, 130.1399, 132.69, 149.2400, 145.7300, 151.69]
[16.4499, 16.5300, 16.9399, 18.09, 23.2800, 39.4300, 40.25, 39.1700, 40.8799, 41.6700, 41.7400, 46.0100, 53.5499, 62.1400, 60.9500, 60.4300, 60.1899, 63.7899, 63.7300, 67.8499, 73.5500, 77.1500, 100.5800, 101.64, 99.2899, 101.41, 99.7100, 101.81, 102.66, 102.19, 106.69, 108.7700, 110.2899, 120.38, 133.9200, 129.66, 136.9499, 138.72, 139.8600, 161.78, 160.7100, 161.8500, 166.2699, 163.0099, 162.16, 167.88, 167.7400, 166.3399, 172.8600, 178.3399, 174.47, 181.0199, 191.4800, 188.0400, 195.4100, 210.6500, 218.4799, 215.8000, 216.3999, 227.13, 223.9499
[48.7400, 47.6299, 50.2800, 49.8400, 51.9399, 55.0799, 69.8400, 71.9799, 71.5399, 71.5500, 73.3400, 74.0500, 73.0099, 84.1500, 112.88, 112.31, 110.92, 112.7399, 113.31, 114.34, 112.9300, 119.62, 122.41, 129.22, 142.2299, 149.8899, 150.4000, 168.4799, 169.2699, 171.5, 171.63, 173.09, 177.19, 180.72, 178.5899, 178.9499, 184.4599, 186.22, 190.4499, 204.5200, 216.84, 223.7900, 226.8600, 227.38, 221.7900, 228.09, 226.38, 221.72, 229.5, 238.97, 243.1700, 240.3600, 247.4499, 246.4800, 254.9900, 270.0900, 270.8500, 272.8600, 285.8600, 280.9199, 286.3400
Explosiveness:
[0.2, 0.2100, 0.2200, 0.2300, 0.2400, 0.2500, 0.2600, 0.2700, 0.2800, 0.2900, 0.3000, 0.3100, 0.3200, 0.3300, 0.3400, 0.3500, 0.3600, 0.3700, 0.3800, 0.3900, 0.4000, 0.4100, 0.4200, 0.4300, 0.4400, 0.4500, 0.4600, 0.4700, 0.4800, 0.4900, 0.5000, 0.5100, 0.5200, 0.5300, 0.5400, 0.5500, 0.5600, 0.5700, 0.5800, 0.5900, 0.6000, 0.6100, 0.6200, 0.6300, 0.6400, 0.6500, 0.6600, 0.6700, 0.6800, 0.6900, 0.7000, 0.7100, 0.7200, 0.7300, 0.7400, 0.7500, 0.7600, 0.7700, 0.7800, 0.7900, 0.8]

Explosiveness: 0.2/0.4/0.6/0.8
Data Type: Trust Count
[2.6, 4.8, 9.0, 10.7, 16.8, 24.1, 29.4, 28.9, 40.4, 40.3, 50.3, 61.1, 66.9, 69.4, 80.5, 92.2, 91.4, 110.0, 123.8]
[6.7, 14.8, 23.6, 37.1, 47.0, 49.3, 64.3, 78.8, 94.7, 101.8, 116.4, 138.5, 157.4, 181.2, 180.9, 192.3, 204.8, 230.5, 234.4]
[21.3, 36.9, 54.2, 68.2, 102.2, 118.3, 142.9, 167.0, 177.6, 206.8, 240.8, 255.6, 273.2, 290.3, 303.5, 336.7, 365.3, 362.7, 383.0]
[28.0, 54.0, 83.2, 108.6, 135.3, 162.0, 176.9, 206.3, 250.2, 259.6, 288.1, 295.7, 375.3, 399.2, 348.3, 389.4, 409.3, 409.7, 420.7]
Average Personal Believe Rate:
[0.05, 0.1, 0.1500, 0.2, 0.25, 0.3, 0.35, 0.3999, 0.4499, 0.4999, 0.5499, 0.6, 0.65, 0.7000, 0.7500, 0.8000, 0.8500, 0.9000, 0.9500

Appendix B (code)

```
import networkx as nx
#import matplotlib.pyplot as plt
import random
import numpy
import math
#import multiprocessing
print 'note: 25000-30000'
n = 1000              #Population Count
m = 30000             #Relationship Count
time = 0.08
interval = 0.01
repostRate = 10    #%%%%%%%%
explosiveness = 0.5
totalTime = 0.1

def custumInt(fl):
    if fl-int(fl)<0.5:
        return int(fl)
    else:
        return int(fl)+1

def negativeExpo(time):
    return math.e**(-2.575*time)

class Person:
    def __init__(self,info,count,spreadCount=0,spreading=False,repost=False):
        self.info = info
        self.count = count
        self.access = False
        self.repost = False
        self.repostrate = repostRate*1.0/100*explosiveness
        self.spreading = spreading
        self.spreadCount = spreadCount
        self.reposted = False

    def calcPost(self,timeline):
        self.repost = self.rePostJudge(repostRate,timeline)

    def rePostJudge(self,rate,timeline):
        p = 1.0 * rate / 100
        self.repostrate=p*negativeExpo(timeline)*explosiveness
        #print self.repostrate
        #print self.repostrate
        if random.random()<self.repostrate:
            return True
        else:
            return False

class Crowd:
    def __init__(self,n,m,time):
        self.dg,self.edgebox = self.creatRelationshipMap(n,m)
        self.nodecolor = []
        self.fontcolor = []
        for i in self.dg:
            self.dg.node[i] = Person(False,0)
            self.nodecolor.append((1,0,0))
        #self.linkrecord = [self.initwithMostSocial()]
        self.initwithMostSocial()
        #self.initwithOne(n)
        self.time = time
```

```
        self.record = [(self.FirstGuy,0)]
        self.nodecolor[self.FirstGuy-1]=(0,0,0.5)
        self.timeLine = 0
        self.i = []


    def initwithMostSocial(self):
        maxcount = 0
        maxindex = 0
        for n in self.dg:
            if len(self.dg[n])>maxcount:
                maxcount = len(self.dg[n])
                maxindex = n
        self.FirstGuy = maxindex
        self.dg.node[self.FirstGuy] = Person(True,0,time,True,True)
        self.dg.node[self.FirstGuy].repost = True
        return maxcount

    def initwithOne(self,n):
        self.FirstGuy = 1
        self.dg.node[self.FirstGuy] = Person(True,0)
        self.dg.node[self.FirstGuy].repost = True
    def getMap(self):
        return self.dg


    def creatRelationshipMap(self,n,m):
        dg = nx.DiGraph()
        nbox = []
        for i in range(n):
            nbox.append(i+1)
        edgebox = []
        dg.add_nodes_from(nbox)
        for i in range(n):
            box = []
            for j in range(int(m/n)):
                randn = random.randint(0,n-1)
                while randn==i or (randn in box):
                    randn = random.randint(0,n-1)
                edgebox.append((i,randn))
                box.append(randn)
            '''
            a = random.randint(1,n)
            b = random.randint(1,n)
            while ((a,b) in edgebox) or a==b:
                a = random.randint(1,n)
                b = random.randint(1,n)
            edgebox.append((a,b))
            '''
        dg.add_edges_from(edgebox)
        return dg,edgebox

    def gammaFunc(self):
        randomVar = numpy.random.gamma(self.time*100, scale=1.0/100, size=None)
        return randomVar

    def update(self):
        self.timeLine += interval
        #print self.timeLine
        icount = 0
        for n in self.dg:
```

```
                if self.dg.node[n].info == True and self.dg.node[n].repost == True:
                    nodes = self.dg[n]
                    nodesCount = len(nodes)
                    repostCount = custumInt(self.dg.node[n].repostrate*nodesCount)
                    #print repostCount
                    repostIndexBox = []
                    for i in range(repostCount):
                        randn = random.randint(0,nodesCount-1)
                        while randn in repostIndexBox:
                            randn = random.randint(0,nodesCount-1)
                        repostIndexBox.append(randn)
                    index = 0
                    box1 = []
                    if not self.dg.node[n].reposted:
                        self.dg.node[n].reposted = True
                        for child in nodes:
                            if self.dg.node[child].info == True:
                                continue
                            else:
                                if self.dg.node[child].access == False:
                                    self.dg.node[child].count = self.time
                                    self.dg.node[child].access = True
                                if index in repostIndexBox:
                                    self.dg.node[child].repost = True
                                index += 1
                    spreading = False
                    for child in nodes:
                        if self.dg.node[child].access == True and self.dg.node[child].info == False:
                            spreading = True
                            break
                    self.dg.node[n].spreading = spreading
                    if spreading:
                        icount += 1
                else:
                    if self.dg.node[n].access == True and self.dg.node[n].info == False and
self.dg.node[n].count<0:
                        self.dg.node[n].info = True

                        self.dg.node[n].rePostJudge(repostRate,self.timeLine)
                        self.record.append((n,self.timeLine,self.dg.node[n].repost,self.dg.node[n].repostrate))
                        #self.nodecolor[n-
1]=((self.timeLine/totalTime)**(1.0/3),(self.timeLine/totalTime)**(1.0/4),0.5+0.5*(self.timeLine/totalTime)**(
1.0/5))
                        #self.linkrecord.append(len(self.dg[n]))
                    elif self.dg.node[n].access == True and self.dg.node[n].count>0:
                        self.dg.node[n].count -= interval
            self.i.append(icount)
            return

        def updateWithTime(self,days):
            ct = int(days/interval)
            for i in range(ct):
                self.update()


import numpy
import math
import random
import networkx as nx
import copy
import matplotlib.pyplot as plt
```

```
tvdata = [0,0,0,0,0,0,9,63,85,90,95,97,98,98,98,98,98,98,98,98,98,98]
radiodata = [0,20,39,55,73,82,80,91,94,94,95,99,99,99,99,99,99,99,99,99,99,99]
yearsdata =
[1920,1925,1930,1935,1940,1945,1950,1955,1960,1965,1970,1975,1980,1985,1990,1995,2000,2005,2010,2011
,2012,2013,2014]
internetdata = [0,0,0,0,0,0,0,0,0,0,0,0,0,0.8,9.2,43.1,68.0,71.7,69.7,79.3,84.2,87.4]
newspaperdata =[109.90184324158706, 122.45127616294623, 135.29367795575217, 126.23338096668866,
130.92343894289897, 145.57656127924932, 148.7016534119077, 142.29382454163897,
137.20741052347248, 130.8302448582204, 127.4612907946311, 118.28779924660442, 114.95516483268226,
110.83677952700845, 104.9032884174072, 93.1205015131286, 86.80000000006874, 75.91375002826089,
59.05552451777496, 59.95106320184407, 59.358991191811846, 60.637578515055914, 58.201582314251176]

for i in range(2015,2051):
    yearsdata.append(i)
    tvdata.append(98)
    radiodata.append(99)
    newspaperdata.append(2978.4369263956787 - 1.448409993618103*i)
    internetdata.append(99.83367 - 66.6891*math.e**(-0.0996124*i + 199.0255752))
total = 1000
internetDelay = 0.08
timeInterval = 0.01
rePostRate = 0.3
explosiveness = 0.5
newspaperDelay = 0.7
totalTime = 3
klink = 25
randlink = klink*40+200
radioDens = 5
radioTrust = 0.1
alpha = 0.5
radioDelay = 0.5
TVDelay = 0.5
TVDens = 20
TVTrust = 1.0/TVDens*0.8
convience = {'newspaper':1,'radio':0.6,'TV':0.8,'internet':0.4}

print 'note: '
print 'klink: '+str(klink)

def produceMediaDictionary(population,rnewspaper,rradio,rtv,rinternet):
    if rnewspaper > 100:
        rnewspaper = 100
    nradio = int(population*(100- rradio))/100
    ntv = int(population*(100-rtv))/100
    ninternet = int(population*(100-rinternet))/100
    nnewspaper = int(population*(100-rnewspaper))/100
    aradio = range(0,population)
    atv = range(0,population)
    ainterest = range(0,population)
    anewspaper = range(0,population)
    for i in range(0,nradio):
        del aradio[random.randint(0,len(aradio)-1)]
    for i in range(0,ntv):
        del atv[random.randint(0,len(atv)-1)]
    for i in range(0,ninternet):
        del ainterest[random.randint(0,len(ainterest)-1)]
    for i in range(0,nnewspaper):
        del anewspaper[random.randint(0,len(anewspaper)-1)]
    aaradio = []
    aatv = []
    aainterest = []
```

```
    aanewspaper = []
    for i in range(0,population):
        if i in aradio:
            aaradio.append(True)
        else:
            aaradio.append(False)
    for i in range(0,population):
        if i in atv:
            aatv.append(True)
        else:
            aatv.append(False)
    for i in range(0,population):
        if i in ainterest:
            aainterest.append(True)
        else:
            aainterest.append(False)
    for i in range(0,population):
        if i in anewspaper:
            aanewspaper.append(True)
        else:
            aanewspaper.append(False)

    for i in range(0,population):
        if aanewspaper[i]:
            if aainterest[i]:
                aatv[i] = False
                aaradio[i] = False
            else:
                if aatv[i]:
                    aaradio[i] = False

        else:
            if aainterest[i]:
                if aatv[i]:
                    aaradio[i] = False
    return (aanewspaper,aaradio,aatv,aainterest)

def customInt(fl):
    if fl-int(fl)<0.5:
        return int(fl)
    else:
        return int(fl)+1

def negativeExpo(time):
    return math.e**(-2.575*time)

def judgeWithRate(r):
    if random.random()<r:
        return True
    else:
        return False

def gammaFunc(value,rate):
        randomVar = numpy.random.gamma(value*rate, scale=1.0/rate, size=None)
        return randomVar

class Person:
    def
__init__(self,info,count,access=False,inInternetSystem=False,inNewspaperSystem=False,inRadioSystem=False,
inTVSystem=False,believe=0.5):
        self.info = info
```

```
            self.count = count
            self.access = access
            self.repost = False
            self.inInternetSystem = inInternetSystem
            self.inNewspaperSystem = inNewspaperSystem
            self.inRadioSystem = inRadioSystem
            self.inTVSystem = inTVSystem
            self.repostrate = rePostRate*explosiveness
            self.reposted = False
            self.believe = believe
            self.infoCount = 0
            self.spreading = False

        def calcRepostProb(self,rate,time):
            p = rate
            self.repostrate=p*negativeExpo(time)*explosiveness

class TVMOD:
        def __init__(self,dg):
            copy_dg = copy.deepcopy(dg)
            self.dg = copy_dg
            self.timeLine = 0
            self.record = []
            self.timerec = []
            self.edgeCount = 0
            for i in range(1000):
                self.dg.node[i].count = gammaFunc(TVDelay,30)
                self.dg.node[i].access = True
                if self.dg.node[i].inTVSystem:
                    self.edgeCount += 20
        def update(self):
            self.timeLine += timeInterval
            for i in range(total):
                if self.dg.node[i].info or self.dg.node[i].inTVSystem == False:
                    continue
                self.dg.node[i].count -= timeInterval
                if self.dg.node[i].count<=0:
                    rate = 1-(1-explosiveness*TVTrust)**TVDens
                    if judgeWithRate(rate):
                        self.dg.node[i].info=True
                        self.record.append(i)
                        self.timerec.append(self.timeLine)
                    else:
                        self.dg.node[i].inTVSystem = False

        def updateWithTime(self,time):
            count = int(time/timeInterval)
            for i in range(count):
                self.update()

        def getResult(self):
            result = {}
            self.updateWithTime(totalTime)
            for i in range(len(self.record)):
                result[self.record[i]] =
{'time':self.timerec[i],'trust':self.dg.node[self.record[i]].believe*convience['TV']}
            for i in range(1000):
                if not (i in result):
                    result[i] = {'time':None,'trust':0}
            return result
```

```
class radioMOD:
    def __init__(self,dg):
        copy_dg = copy.deepcopy(dg)
        self.dg = copy_dg
        self.timeLine = 0
        self.record = []
        self.timerec = []
        self.edgeCount = 0
        for i in range(1000):
            self.dg.node[i].count = gammaFunc(radioDelay,30)
            self.dg.node[i].access = True
            if self.dg.node[i].inRadioSystem:
                self.edgeCount += 5

    def update(self):
        self.timeLine += timeInterval
        for i in range(total):
            if self.dg.node[i].info or self.dg.node[i].inRadioSystem == False:
                continue
            self.dg.node[i].count -= timeInterval
            if self.dg.node[i].count<=0:
                rate = 1-(1-explosiveness*radioTrust)**radioDens
                if judgeWithRate(rate):
                    self.dg.node[i].info=True
                    self.record.append(i)
                    self.timerec.append(self.timeLine)
                else:
                    self.dg.node[i].inRadioSystem = False

    def updateWithTime(self,time):
        count = int(time/timeInterval)
        for i in range(count):
            self.update()

    def getResult(self):
        result = {}
        self.updateWithTime(totalTime)
        for i in range(len(self.record)):
            result[self.record[i]] =
{'time':self.timerec[i],'trust':self.dg.node[self.record[i]].believe*convience['radio']}
        for i in range(1000):
            if not (i in result):
                result[i] = {'time':None,'trust':0}
        return result


class newspaperMOD:
    def __init__(self,dg):
        copy_dg = copy.deepcopy(dg)
        self.dg = copy_dg
        self.timeLine = 0
        self.record = []
        self.timerec = []
        self.edgeCount = 0
        edgebox = []
        for i in range(total/50):
            lcstr = 'local-'+str(i)
            self.dg.add_node(lcstr)
            for j in range(total/20):
                index = j+i*50
                edgebox.append((lcstr,index,gammaFunc(newspaperDelay,40)))
```

```
                    if self.dg.node[index].inNewspaperSystem:
                        self.edgeCount += 1
            self.dg.add_weighted_edges_from(edgebox)
            for i in range(20):
                lcstr = 'local-'+str(i)
                for j in range(50):
                    index = i*50 + j
                    self.dg.node[index].count = self.dg.edge[lcstr][index]['weight']
                    self.dg.node[index].access = True

        def update(self):
            self.timeLine += timeInterval
            for i in range(total):
                if self.dg.node[i].info or self.dg.node[i].inNewspaperSystem == False:
                    continue
                self.dg.node[i].count -= timeInterval
                if self.dg.node[i].count<=0:
                    if judgeWithRate(explosiveness/10):
                        self.dg.node[i].info=True
                        self.record.append(i)
                        self.timerec.append(self.timeLine)
                    else:
                        self.dg.node[i].inNewspaperSystem = False

        def updateWithTime(self,time):
            count = int(time/timeInterval)
            for i in range(count):
                self.update()

        def getResult(self):
            result = {}
            self.updateWithTime(totalTime)
            for i in range(len(self.record)):
                result[self.record[i]] =
{'time':self.timerec[i],'trust':self.dg.node[self.record[i]].believe*convience['newspaper']}
            for i in range(1000):
                if not (i in result):
                    result[i] = {'time':None,'trust':0}
            return result

class internetMOD:
        def __init__(self,dg,initialQuan):
            copy_dg = copy.deepcopy(dg)
            self.edgeCount = 0
            self.dg = self.creatLittleWorldEdges(copy_dg)
            self.timeLine = 0
            self.record = []
            self.timerec = []
            initialGuys = []
            totalInSys = 0
            for n in dg:
                if self.dg.node[n].inInternetSystem:
                    totalInSys += 1
            if initialQuan>totalInSys:
                initialQuan = totalInSys
            for i in range(initialQuan):
                randn = random.randint(0,999)
                while not self.dg.node[randn].inInternetSystem or randn in initialGuys:
                    randn = random.randint(0,999)
                initialGuys.append(randn)
            for n in initialGuys:
```

```
            self.dg.node[n].info = True
            self.dg.node[n].repost = True
            self.record.append(n)
            self.timerec.append(0)
            self.dg.node[n].infoCount += 1
            self.dg.node[n].spreading = True
        self.spreadCount = []

    def creatLittleWorldEdges(self,dg):
        fullbox = []
        edgebox = []
        for i in range(50):
            fullbox.append(i)
        for reg in range(20):
            for j in range(50):
                index1 = reg*50+j
                chosen = []
                for k in range(klink):
                    randn = random.randint(0,49)
                    while (randn in chosen) or (reg*50+randn==index1):
                        randn = random.randint(0,49)
                    chosen.append(randn)
                for k in range(klink):
                    index2 = reg*50+chosen[k]
                    edgebox.append((index1,index2))
                    if dg.node[index1].inInternetSystem and dg.node[index2].inInternetSystem:
                        self.edgeCount += 1
        dg.add_edges_from(edgebox)
        edgebox = []
        for i in range(randlink):
            a = random.randint(0,999)
            b = random.randint(0,999)
            areg = a/50
            breg = b/50
            while a==b or areg==breg or ((a,b) in edgebox):
                a = random.randint(0,999)
                b = random.randint(0,999)
                areg = a/50
                breg = b/50
            edgebox.append((a,b))
            if dg.node[a].inInternetSystem and dg.node[b].inInternetSystem:
                self.edgeCount += 1
        dg.add_edges_from(edgebox)
        return dg

    def update(self):
        self.timeLine += timeInterval
        spreadCt = 0
        for n in self.dg:
            if self.dg.node[n].inInternetSystem == False:
                continue
            if self.dg.node[n].info == True and self.dg.node[n].repost == True:
                nodes = self.dg[n]
                nodesCount = len(nodes)
                repostCount = customInt(self.dg.node[n].repostrate*nodesCount)
                repostIndexBox = []
                for i in range(repostCount):
                    randn = random.randint(0,nodesCount-1)
                    while randn in repostIndexBox:
                        randn = random.randint(0,nodesCount-1)
                    repostIndexBox.append(randn)
```

```
                    index = 0
                    spread = False
                    if not self.dg.node[n].reposted:
                        self.dg.node[n].reposted = True
                        for child in nodes:
                            self.dg.node[child].infoCount += 1
                            if self.dg.node[child].inInternetSystem == False or self.dg.node[child].info ==
True:
                                continue
                            else:
                                spread = True
                                if self.dg.node[child].access == False:
                                    self.dg.node[child].count = internetDelay
                                    self.dg.node[child].access = True
                                if index in repostIndexBox:
                                    self.dg.node[child].repost = True
                            index += 1
                    else:
                        for child in nodes:
                            if not (self.dg.node[child].inInternetSystem):
                                continue
                            elif self.dg.node[child].access == True and self.dg.node[child].info == False:
                                spread = True
                                break
                    if spread:
                        spreadCt += 1
                else:
                    if self.dg.node[n].access == True and self.dg.node[n].info == False and
self.dg.node[n].count < 0:
                        self.dg.node[n].info = True
                        self.dg.node[n].calcRepostProb(rePostRate,self.timeLine)
                        self.record.append(n)
                        self.timerec.append(self.timeLine)
                    elif self.dg.node[n].access == True and self.dg.node[n].count > 0:
                        self.dg.node[n].count -= timeInterval
            self.spreadCount.append(spreadCt)
            return

        def howManyWorldKnows(self):
            regCount = 0
            for reg in range(20):
                for j in range(50):
                    index = reg*50+j
                    if self.dg.node[index].info == True:
                        regCount += 1
                        break
            return regCount

        def updateWithTime(self,days):
            ct = int(days/timeInterval)
            for i in range(ct):
                self.update()

        def getResult(self):
            result = {}
            self.updateWithTime(totalTime)
            for i in range(len(self.record)):
                result[self.record[i]] =
{'time':self.timerec[i],'trust':self.dg.node[self.record[i]].believe*self.dg.node[self.record[i]].infoCount*convienc
e['internet']}
            for i in range(1000):
```

```
                if not (i in result):
                    result[i] = {'time':None,'trust':0}
            return result
class Crowd:
    def __init__(self,newspaperRate,radioRate,TVRate,internetRate,netInitial = 1):
            calculatedRates = produceMediaDictionary(total,newspaperRate,radioRate,TVRate,internetRate)
            self.newspaperBox = calculatedRates[0]
            self.radioBox = calculatedRates[1]
            self.TVBox = calculatedRates[2]
            self.internetBox = calculatedRates[3]
            self.dg = self.creatSmallWorld(randlink)
            self.netInitial = netInitial

    def creatSmallWorld(self,randlink):
            dg = nx.DiGraph()
            node = []
            for i in range(1000):
                node.append(i)
            dg.add_nodes_from(node)
            for i in range(1000):
                dg.node[i] =
Person(False,None,inNewspaperSystem=self.newspaperBox[i],inRadioSystem=self.radioBox[i],inTVSystem=s
elf.TVBox[i],inInternetSystem=self.internetBox[i],believe=numpy.random.beta(alpha,1-alpha))
            edgebox = []
            fullbox = []
            return dg

    def getMin(self,tup):
            Min = None
            for i in tup:
                if i == None:
                    continue
                elif Min == None:
                    Min = i
                elif i < Min:
                    Min = i
            return Min

    def combineResults(self):
            myNews = newspaperMOD(self.dg)
            myRadio = radioMOD(self.dg)
            myTV = TVMOD(self.dg)
            myInternet = internetMOD(self.dg,self.netInitial)
            newsResult = myNews.getResult()
            radioResult = myRadio.getResult()
            TVResult = myTV.getResult()
            internetResult = myInternet.getResult()
            finalResult = {}
            for key in range(1000):
                eachTime =
self.getMin((newsResult[key]['time'],radioResult[key]['time'],TVResult[key]['time'],internetResult[key]['time']))
                eachTrust =
newsResult[key]['trust']+radioResult[key]['trust']+TVResult[key]['trust']+internetResult[key]['trust']
                finalResult[key] = {'time':eachTime,'trust':eachTrust}
            return finalResult
```