Crime Detecting System
A Powerful App. to Unveil Black Sheep

Summary

In this article we design three models to judge whether a person involves in a criminal case. Meanwhile, they are applied in more general cases, even in other fields.

Firstly, CrimeRank Model is designed to judge if a particular person is a conspirator at a given confidence level. This algorithm is mainly based on social network system, which emphasizes neighbors' influence on each person.

Secondly, CrimeSource Model is built based on source fields principles in physics to detect potential conspirators. Some particular people are regarded as sources that emit suspicious messages. By simulating the physics process we reach final list of suspects.

Thirdly, Decoder Model is contrived to analyze the relationship between a certain topic and the criminal case. We endow every notional word a value to measure its significance. Then we calculate the importance of a topic by weighing the value of words in it.

Afterwards, we devised Modified CrimeRank Model by applying Decoder Model to the original one. Thus the degree of suspicious of each topic could be automatically calculated. The final list of suspects might be more convincing.

Finally, we propose some suggestions for further research of these three models. An amazing result could be reached by applying our model in international relationship network during Cold War period.

Content

Introduction

In a great number of criminal cases, evidences are usually incomplete and vague at the beginning of investigation. Thereupon quite a few technologies have been designed to sniff out the potential conspirators, which, to a certain extent, reduce the difficulty to seize real criminals.

Entity Extraction (M. Chao, 2002) identifies particular patterns from data such as messages, phone-call record or images, which automatically distinguish personal characteristics from their reports to police. *Clustering Techniques* (R. V. Hauck, 2002) categorizes data into several groups which maximize intraclass similarity. For example, this technique helps investigators to group a large number of people who share the same habits. *Deviation Detection* is a method to seek out the unusual text messages or video records that differs markedly from other data. *Social Network Analysis* describes each person's role in a whole network system. At the same time, the flow of tangible or intangible data flow can be traced.

In the following passages, we will develop several models to detect the possibility of each person's involvement in a crime. With powerful tools of data-digging and text-analysis, original data could be put into algorithm to accurately find out latent conspirators. Our model will be later modified to suit more general situations, to meet the need of various sorts of information containing in a crime and different initial states. Furthermore, the modified model could have amazing applications in fields other than crime detection. We will discuss such kind of utilization in a certain field.

Background

In a company some people have been considered involved in a crime. Several employees have been confirmed to be conspirators while several others innocent. All staff communicated with each other on certain topics. The ICM (Intergalactic Crime Modelers) thought some topics are related to the crime. We want to find out whether a person is a conspirator based on given communication records and other information.

Assumptions

(1) Everyone could be a suspect unless he/she is convinced innocent.

(2) There exists possibility of arbitrary pairs of people to exchange messages that may not be recorded by ICM.

(3) People cannot be influenced by messages from outside the company. That is to say, we can only reach conclusions through information inside the company.

(4) The given database of messages is free from suspicion and its reliability is guaranteed by people recording it.

(5) The known conspirators and innocents remain their identities throughout the investigation. This could not be challenged by any subsequent procedure.

CrimeRank Model

In PageRank Model (Brin S, Page L, 1998), each webpage is given the same initial endowment PageRank (*Pr*), the chance that a user will stay on this page. Thereafter, if both page B and C are linked to page A, the *Pr* of B and C will gather to A. We have

*Pr(*A*)=Pr(*B*)+Pr(C)*.

Enlightened by the above-mentioned idea, we establish our model named after CrimeRank to find out the likelihood of being conspirators. The odds of involving in the crime of person A is noted by *Cr*(A). Based on given information we endow initial values to everyone inside the company. The *Cr* value of a particular person will be influenced by his/her neighbors who convey messages to him.

$$Cr(\mathrm{Pi}) = \sum_{pj \in M(Pi)} Cr(Pj) \tag{1}$$

Where *Pi* refers to i[th] person, *M(Pi)* is the set of people who directly communicate topics to *Pi*.

Suppose B communicates topics to A and C, while D communicates message not only to A but to four other persons. The contributed factor from B to A has to be divided by the total number of topics sent from B to recipients. So it is with everyone sending messages to A.

$$Cr(Pi) = \sum_{pj \in M(Pi)} \frac{Cr(Pj)}{L(Pj)} \tag{2}$$

Where *L(Pj)* refers to the total number of topics given by *P*j to all recipients.

However, equation (2) fails to distinguish the suspicion-known topics from all other suspicion-unidentified topics. All topics are composed of suspicion-known topics and suspicion-unidentified ones. In fact, suspicion-unidentified topics could be suspicious. As a result, an influence factor α should be taken into account to emphasize the suspicious-known topics.

$$Cr(P_i) = \sum_{pj \in M(Pi)} \frac{Cr(Pj)}{L(Pj)} * (L_{sus-known}(Pj \to Pi) + \alpha L_{sus-unidentified}(Pj \to Pi)) \quad (3)$$

Where *Pi* refers to i[th] person; *M(Pi)* is the set of people who directly communicate topics to him; *L(Pj)* refers to the total number of topics given by *P*j to all recipients; $L_{sus-known}(Pj \to Pi)$ is the number of suspicious-known topics from *P*j to *Pi* while $L_{sus-unidentified}(Pj \to Pi)$ is the remaining suspicious-unidentified topics from *P*j to *P*i; α is a coefficient ranging in (0,1] to weaken the impact of suspicious-unidentified topics.

Damping Factor

The previous model managed to include other people's influence on the chance of committing a crime but neglected the intrinsic impulse to misbehave. Hence, we should take other factors into the CrimeRank Model.

$$Cr(P_i) = \frac{1-d}{N} + d \sum_{pj \in M(Pi)} \frac{Cr(Pj)}{L(Pj)} * (L_{sus-known}(Pj \to Pi) + \alpha L_{sus-unidentified}(Pj \to Pi)) \quad (*)$$

Where N is the total number of staff; d is the damping factor to balance intrinsic tendency and extrinsic influence from other people; *Pi* refers to i[th] person; *M(Pj)* is the set of people who communicate topics to him; *L(Pj)* refers to the total number of topics sent by *P*j to all recipients; $L_{sus-known}(Pj \to Pi)$ is the number of suspicious-known topics from *P*j to *P*i while $L_{sus-unidentified}(Pj \to Pi)$ is the remaining suspicious-unidentified topics from *P*j to *P*i; α is a coefficient ranging in (0,1] to weaken the impact of suspicious-unidentified topics.

Notice equation (*) is our final version of CrimeRank Model.

According to *Crime Analysis for Problem Solvers* (Ronald V. Clarke & John E. Eck,

2005), in crime 20 percent of some things are responsible for 80 percent of the outcomes. Therefore, in our model, 20 percent of *Cr* can be traced back to one's intrinsic aptitude. As a result, we assume d=0.8. Yet it is not necessarily the case, this assignment gives us an illuminating and powerful tool to describe the situation.

Notice that α is a coefficient ranging in (0,1] to weaken the impact of suspicious-unidentified messages, which measures the relative importance of suspicious-unidentified topics to suspicious-known ones. According to *Gossip in Evolutionary Perspective* (R. I. M. Dunbar, *Modern Psychology Research*, 2004), 78 percent of topics in interpersonal communication are on social issues which have nothing to do with business. As a result, we assume $\alpha = 0.22$. That is to say, a suspicion-unidentified message topic has 22 percent possibilities to be actually suspicious.

In this case, there are three kinds of people: known conspirators, known innocents and all the rest people open to be suspected. Our task is to acquire the reliable *Cr*(Pi) to distinguish conspirators from all staff.

Now we start to solve the CrimeRank Model. At the very beginning, without the loss of generality, we assume $\sum_{i} Cr(Pi) = 1$. We should arrange suitable initial *Cr* value to everyone to show that some people are affirmed to be guilty while some others innocent. To be fair enough, we find it a reasonable settlement to make anyone who is surely to be a conspirator get the initial $Cr = \frac{1}{2 * number\ \ of\ \ conspirators}$ while anyone who is surely to be innocent get the initial *Cr*=0. Then rest people get the initial

$$C\mathrm{r} = \frac{1}{2(\textit{total number of people} - \textit{number of conspirators} - \textit{number of nonconspirators})}.$$

In the following passage, equation (*) will be used to accomplish the Flow Graph to unveil the final result.

Flow Graph

Now we use Microsoft Visual C++ to complete Flow Graph.

(1) Consider every person as a vertex. When Pi sends a message to *P*j, an edge will be built from *P*i to *P*j. Then we get a directed graph G.

(2) Endow each vertex an initial Cr value with the foregoing rule.

(3) Calculate the new *Cr*(Pi) (i from 1 to n, n is the total number of nodes) by applying CrimeRank Model Equation (*).

(4) Repeat procedure (3) until *Cr* value of each nodes remains stable. That is to say, each time we finish doing procedure (3), we will examine whether the new $\mathrm{Cr}(\mathrm{Pi})^{(K+1)}$ (K+1 means *Cr* value after K+1 times iterations) is close enough to $Cr(Pi)^{(k)}$. Finally we can reach the situation $\left|\mathrm{Cr}(\mathrm{Pi})^{(K+1)} - \mathrm{Cr}(\mathrm{Pi})^{(K)}\right| < \varepsilon$ (we pick $\varepsilon = 0.0000001$) for every *Pi*, which indicates all *Cr* values are stable.

(5) Now we get the final result of *Cr*. Here is an additional step to normalize *Cr*(Pi) to clearly distinguish them. The normalization of *Cr*(Pi) can be described in the following process:

Step One: $Cr^*(Pi) = \dfrac{Cr(Pi) - \overline{Cr(Pi)}}{\sigma(Cr(Pi))}$,

Step Two: $Cr^{**}(Pi) = \int_{-\infty}^{Cr^*(Ti)} \dfrac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, $0 \le Cr^{**}(Pi) \le 1$

According to the *Central Limit Theorem* (CLT), sufficiently large number of independent random variables, each with finite mean and variance, will be approximately normally distributed. Hence Cr^{**} will be asymptotically normal distributed. So we can use Z-test to determine whether a certain person is a conspirator at a given confidence level.

A Simple Example (EZ Investigation)

Here we apply CrimeRank to EZ investigation. The result is shown by Figure 1.

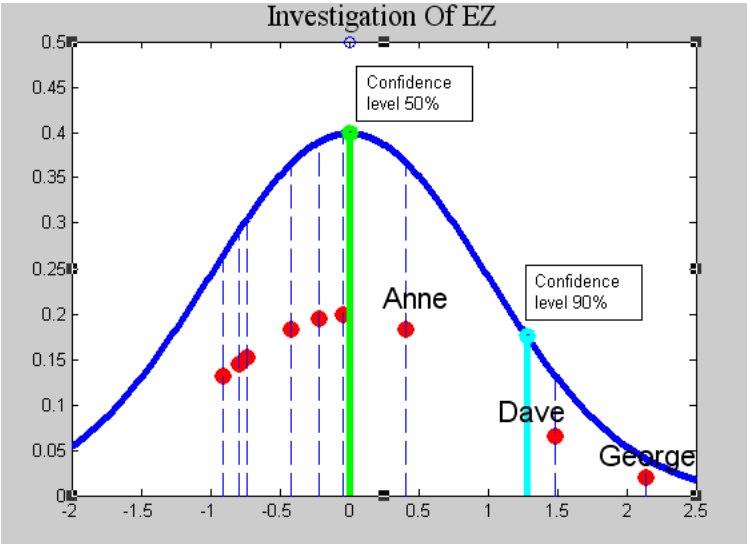

Figure 1

At 90% confidence level, Dave and George are considered conspirators, which is known in advance. So it indicates that the confidence level is selected too high. In order to detect the person who is likely to a conspirator, a 50% confidence level could be chosen. In this situation, George, Dave and Anne are three conspirators according to CrimeRank Model. However, the result does not meet exactly with the real situation. In fact, Bob is a conspirator.

Yet, with the small size of sample and incomplete communication information, it's

hard to acquire the accurate results. We will apply our model in a more complex example.

A Complex Example

Here we apply CrimeRank to a more complex example (**Requirement 1**).

The result is shown by Figure 2.

Figure 2a Normalized CrimeRank Value 1

*Harvey	1.0000	Kristine	0.6444	Lois	0.3690	Bariol	0.2495
*Alex	0.9994	Dwight	0.5859	Gerry	0.3490	Marian	0.2484
$Dolores	0.9990	*Paul	0.5763	Claire	0.3343	Douglas	0.2332
*Yao	0.9969	William	0.5605	Kim	0.3289	Hark	0.2203
$Jerome	0.9379	Patrick	0.5532	Christina	0.3285	Chara	0.2197
#Darlene	0.9157	Beth	0.5327	Melia	0.3241	Cha	0.2098
$Gretchen	0.9106	Beth	0.5307	Mai	0.3185	Andra	0.2098
*Ulf	0.9063	Stephanie	0.5101	Louis	0.3126	Olina	0.2023
Karen	0.8825	Neal	0.5062	Malcolm	0.3101	Darol	0.2023
Neal	0.8796	Francis	0.5041	Quan	0.3082	#Ellin	0.1950
Franklin	0.8490	Priscilla	0.5041	Cory	0.3076	Seeni	0.1706
Fanti	0.8246	Crystal	0.4811	#Tran	0.3067	Dayi	0.1693
Elsie	0.8205	Wayne	0.4724	Marcia	0.3036	Vind	0.1693
Sheng	0.8025	Kristina	0.4185	Hazel	0.2978	Lao	0.1693
#Paige	0.7797	*Jean	0.4174	Cole	0.2957	Lars	0.1693

Donald	0.7763	Eric	0.4154	Han	0.2873	Le	0.1693
Gretchen	0.7445	*Elsie	0.4021	#Chris	0.2737	#Jia	0.1693
Julia	0.7238	Shelley	0.3799	Reni	0.2706	Carina	0.1693
Marion	0.6996	Sandy	0.3783	#Este	0.2619	#Gard	0.1693
Sherri	0.6764	Wesley	0.3759	Erica	0.2568	Phille	0.1693
Patricia	0.6669	Katherine	0.3759	Jerome	0.2529		

* indicates prior known conspirators, # indicates prior known non-conspirators,$ indicates senior managers

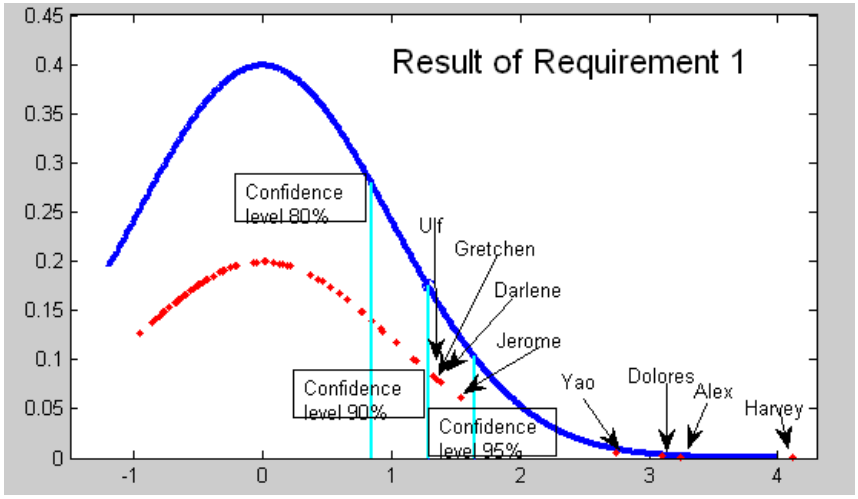

Figure 2b

At 95% confidence level, Harvey, Alex, Dolores, Yao are considered conspirators. Senior manager Dolores is involved in the crime. At 90% confidence level, Harvey, Alex, Dolores, Yao, Jerome, Darlene, Gretchen and Ulf are thought to be involved in the crime. Notice there are two persons name “Jerome” (No.16 & 34) and another two people named “Gretchen” (No.4 & 32). We reach the conclusion at 90% confidence level, senior manager Jerome (No. 34), Gretchen (No. 4) and Dolores are all conspirators. Notice at 90% level, Darlene will be considered a conspirator but in fact he is innocent. It does not indicate the invalidity of the model, because at 90% level, a

10 percent chance of making errors is still allowed.

To solve Requirement 2, what we should only do is to adjust several parameters since our CrimeRank Model is suitable to general condition. For instance, when it comes to light that Topic 1 is suspicious-known, we will turn messages related to Topic 1 into suspicion-known ones and delete them from suspicion-unidentified. At the same time, the initial *Cr* value will also be adjusted because of Chris' emergence in the list of conspirators.

The result of requirement 2 is shown in Figure 3. Details are shown in Appendix I.

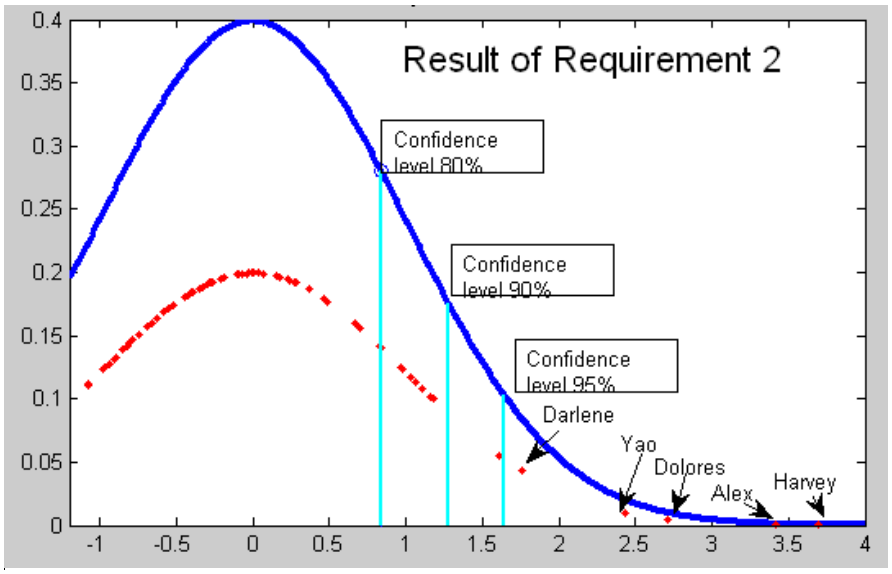

Figure 3

At 90% confidence level, Harvey, Alex, Dolores, Yao, Darlene and Jerome are considered conspirators. Comparing the result in Requirement 1, Ulf and Gretchen are not believed to be conspirators. At this time, only two senior managers: Dolores and Jerome are thought to be conspirators.

CrimeSource Model

In Physics, there are a certain number of sources emitting energy in fields like the

electric or gravitational field. In these fields, the potential energy at a certain position is related to its distance to the source by $E = k\frac{q}{r}$ (E is the potential energy, q is a characteristic value of the source, k is a constant coefficient, and r is the distance).

Kindled by those principles, the known conspirators could be viewed as sources, emitting suspicious messages like positive energy. Meanwhile, the identified non-conspirators are regarded as sinks which absorb messages of suspicion, or sources that send out negative energy to offset the influence of known conspirators. As a person's distance to a known conspirator in the message network decreases, his/her suspicion increases, which could be described as his/her potential energy in the "message field". To calculate that energy, we simulate the equation in source fields principles, giving that $D = k\frac{q}{r}$ (D is the degree to which the person is suspicious/innocent, q is a characteristic value of the source, k is a constant coefficient, and r is the distance). Based on these, 9 steps are designed to determine how a certain person is suspicious in the message network:

1. As stated before, the suspicion-unidentified messages have 22% possibilities to be suspicious. As a result, the number of the real suspicious messages $n(t)$can be calculated as $n(t) = n(s) + 0.22n(u)$. (*n(s)* is the number of the known suspicious messages, and *n(u)* is the number of the suspicion-unidentified messages)
2. $r(d) = \frac{1}{n(t)}$.

   Where $r(d)$ is the direct distance between two people; $n(t)$is the number of the real suspicious messages).

   The more suspicious messages transmitted between two people, the closer their

relationship is in a criminal sense.

3. Some information may be communicated indirectly, so we can draw a path from person A through some middlemen to person B to identify them.

$$r(i) = \sum_j r_j(d)$$

Where $r(i)$ is the indirect distance between person A and B; $r_j(d)$ is the *j*th direct distance on the pass from A to B.

4. $r(t) = \mathrm{Min}\{r(d), r(i)\}$

Where r(t) is the real distance between two people.

5. $SD_{mn} = k\frac{q}{r_{mn}}$

Where SD_{mn} is the suspicion degree of person m for the known conspirator n; q is a characteristic value of all conspirators, k is a constant coefficient, and r_{mn} is the distance between person m and n(m is an unidentified person).

6. $SD_m = \sum_n SD_{mn}$

Where SD_m is he suspicion degree of person m.

7. Similar to the calculation of the suspicion degree, the innocent degree (ID) of m is the sum of ID_{ml}, which is $ID_{ml} = k\frac{q}{r_{ml}}$.

8. $ID_m = \sum_l ID_{ml}$.

Where ID_m is the innocent degree of person m.

9. The combined degree of m^{th} person (CD_m): $CD_m = SD_m - ID_m$.

The CD_m of the employees in the company catering to Requirement 1 are listed in Figure 4, which reflects the suspect level of m^{th} person.

Figure 4

The Combined Degree							
Seeni	101.116957	Marion	14.310521	Julia	5.333733	Olina	0.922202
Elsie	75.689619	Franklin	14.034473	Patricia	4.929538	Le	-0.167192
Dolores	44.928069	Eric	13.361099	Louis	4.520849	Hark	-0.169396
Priscilla	29.098725	Neal	12.98263	Erica	4.442578	Cory	-0.356832
Dwight	28.746199	Katherine	11.924991	Beth	4.299436	Cha	-0.73194
Lars	27.593415	Malcolm	10.540188	Darol	3.865162	Shelley	-0.745408
Sherri	23.718291	Donald	8.344137	Claire	3.651158	Phille	-0.833179
William	23.016938	Christina	8.089879	Marian	3.501287	Sheng	-1.057656
Beth	22.451922	Lois	7.993536	Karen	3.154793	Wayne	-1.590663
Kim	21.890904	Jerome	7.697861	Melia	3.09179	Vind	-1.64985
Stephanie	21.292322	Jerome	7.361138	Chara	2.522202	Cole	-1.886655
Douglas	20.327116	Marcia	7.298972	Gretchen	2.460589	Quan	-2.300526
Neal	20.233822	Han	6.533201	Andra	1.699569	Gerry	-3.288889
Gretchen	16.828507	Hazel	6.497156	Crystal	1.68776	Reni	-3.796007
Dayi	16.737372	Kristina	6.425743	Mai	1.37612	Kristine	-3.806496
Francis	16.286174	Sandy	5.818083	Wesley	1.308919	Fanti	-4.259542
Patrick	15.973465	Carina	5.353468	Lao	0.961064	Bariol	-6.26742

Decoder Model

In CrimeRank Model, some topics are predetermined whether they are suspicious. However, in more general cases, no additional information will be provided whether a certain kind of topic is related to criminal behavior. For example, it seems topic like ”having lunch together” or ”going to supermarket after job” has nothing to do with crime. But under some particular circumstances like a plot to steal food in a restaurant near a supermarket, such information might be taken into account to help discern criminals. Consequently, we plan to design an intellectual system to sniff out potential key information to find criminals. Finally, Decoder Model will be applied to CrimeRank Model to automatically rank the possibility of a person’s involvement in conspiracies.

VSM (*Vector Space Model*，Salton,1971) is generally applied in classifying sentences. Each sentence has several feature terms to describe the main characteristic.

Afterwards VSM multiples term features with term weight to achieve the eigenvalue of a sentence, which could be used to judge the similarity of two sentences or for further usage. Some other scholars raised other theories in text categorization. DTP (*Distance to Transition Point*, Moyotl et al, 2005) and SCIW (*Strong Information Class Word*, Li et al, 2005) are designed to detect the significance of sentences. However, these models are all built at the base of experimental statistics. Because of the difference in words, grammar and classifying tools used, their methods might entail improvement in the future.

In our Decoder Model, we should judge the feature terms in a topic. Notional words in a sentence are treated as feature terms of a sentence. We are here to devise a model to evaluate the "eigenvalue" *EV(Wj)* of each notional word *Wj*, i.e. the importance of message containing in this word. Illuminated by our CrimeRank Model, we regard any notional word as a vertex, omitting empty words like prepositions and conjunctions. Once any two words appear in the same topic, we link an edge between them. Afterwards CrimeRank renders us an algorithm to calculate the "eigenvalue" of a word. Notice in this algorithm we can easily achieve the stable result after certain times of iteration. We endow the initial "eigenvalue" of each notional word to be 1. (In fact, in our algorithm the initial value does not exert influence on the final result .)

Due to the huge number of words involving in those 15 topics, it is hard to graphically show the result. For the convenience of demonstration, here we only list first 10 alphabetical words' eigenvalue on a radar map.

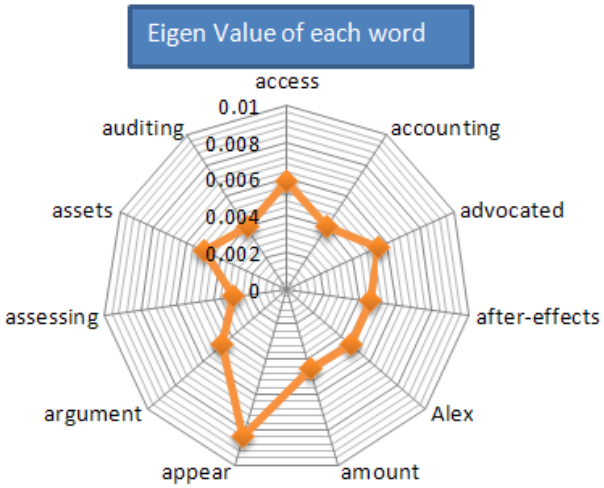

Figure 5

Now we have got the "eigenvalue" of each notional word. (We name them as *EV*(Wi)).We face the problem of how to evaluate the relative significance of each topic. *Term Weight Algorithm* has been proposed to solve this problem by weighed eigenvalue. Mention a few most frequently used methods, such as:

(1) *Boolean Weight* $wij = \begin{cases} 1, & if \quad tfij > 0 \\ 0, & otherwise \end{cases}$ (tf_{ij} is the frequency of word *Wi* appearing in topic Tj).

(2) TF (*Term Frequency*, Gerard Salton, 1983)Weight $w_{ij} = tf_{ij}$.

(3) IDF(*Inverse document frequency*, Spark Jones, 1972) Weight $wij = \log \frac{N}{n_i}$ (N is the number of topics; n_i is the number of topics which contain word *Wi*).

In our Decoder Model, we combine TF weight with IDF weight, which is named after TF-IDF Weight:

$$w_{ij} = tf_{ij} * \log \frac{N}{n_i} \qquad (4)$$

Where tf_{ij} is the relative frequency of word W_i appearing in topic T_j; N is the number of topics; n_i is the number of topics which contain word W_i.

Notice W_{ij} is in proportion to the frequency W_i appears in T_j but in inverse proportion to the frequency of topics which contain word W_i. We think it is a good way to calculate the term weight, because it takes into account both the influence of the frequency of a word in a certain topic and that in all topics.

It is time to give each topic a value (TopicValue, abbr. $TPR(Ti)$,i=1,..15) to assess the impact of a certain topic:

$$TPV(Ti) = \sum_{Wj \in Ti} EV(W_j) * w_{ji} = \sum_{Wj \in Ti} EV(W_j) * tf_{ji} * \log \frac{N}{n_j} \qquad (5)$$

Where $EV(W_j)$ is the eigenvalue of word j; W_{ij} is the TF-IDF weight; tf_{ji} is the frequency of word W_j appearing in topic Ti; N is the number of topics; n_j is the number of topics which contain word W_j.

After the calculation, it has been noticed that the $TPV(Ti)$ is too close to 0 to distinguish the relative importance of topics. So we use normalized $TPV(Ti)$ in further calculation process to avoid this weakness. Notice this method could be used in more general cases to handle with various types of data.

$$\text{Step1}: TPV^*(Ti) = \frac{TPV(Ti) - \overline{TPV(Ti)}}{\sigma(TPV(Ti))}$$

$$\text{Step2}: \quad TPV^{**}(Ti) = \int_{-\infty}^{TPV^*(Ti)} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Notice now TPV^{**} is normalized TopicValue. The result is shown in Figure 6. The larger normalized TopicValue, the more important the topic.

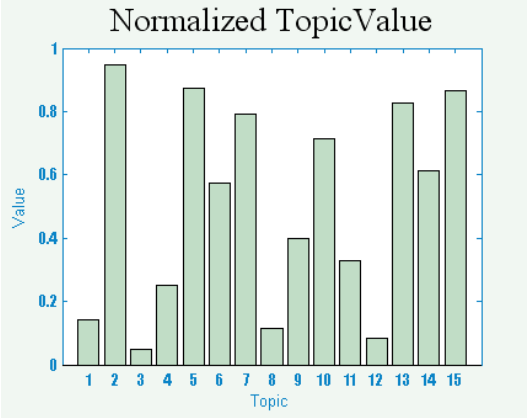

Figure 6

Modified CrimeRank Model with Application of Decoder Model

Decoder Model, a particular form of semantic network analysis method, was put into use to determine the relative importance of a certain topic when cracking a criminal case. We combine Decoder Model with CrimeRank Model, which is designed at the beginning, to comprehensively judge the possibility a person to be a conspirator.

Recall the equation (*) in CrimeRank Model:

$$Cr(P_i) = \frac{1-d}{N} + d \sum_{pj \in M(Pi)} \frac{Cr(Pj)}{L(Pj)} * (L_{sus}(Pj \to Pi) + \alpha L_{unsus}(Pj \to Pi)) \quad (*)$$

α is a factor to weaken the influence of unsuspicious topics. With our Decoder Model, we have got a powerful criterion, $TPR^{**}(0 \le TPR^{**} \le 1)$ to judge the relative influence of each topic. As a result, we can transform equation (*) into the following equation (**), with Decoder Model's application to CrimeRank Model:

$$Cr(P_i) = \frac{1-d}{N} + d \sum_{pj \in M(Pi)} \frac{Cr(Pj)}{L(Pj)} * (\sum_k TPV^{**}(T_k) L_{Tk}(Pj \to Pi)) \quad (**)$$

Where N is the total number of staff; d is the damping factor to balance intrinsic tendency and extrinsic influence from other people to commit a crime; P_i refers to i^{th}

person; $M(P_i)$ is the set of people who communicate topics to him; $L(P_j)$ refers to the total number of topics given by Pj to all recipients; $TPV^{**}(T_k)$ is the TPV^{**} value of the k[th] topic and $L_{Tk}(Pj \to Pi) = \begin{cases} 1, when\ Pj\ communicates\ topic\ Tk\ to\ Pi \\ 0, when\ Pj\ doesnot\ communicate\ topic\ Tk\ to\ Pi \end{cases}$

With the same procedure in CrimeRank Model, we come to the final result of suspicion list of the crime (**under the condition of requirement 1**)

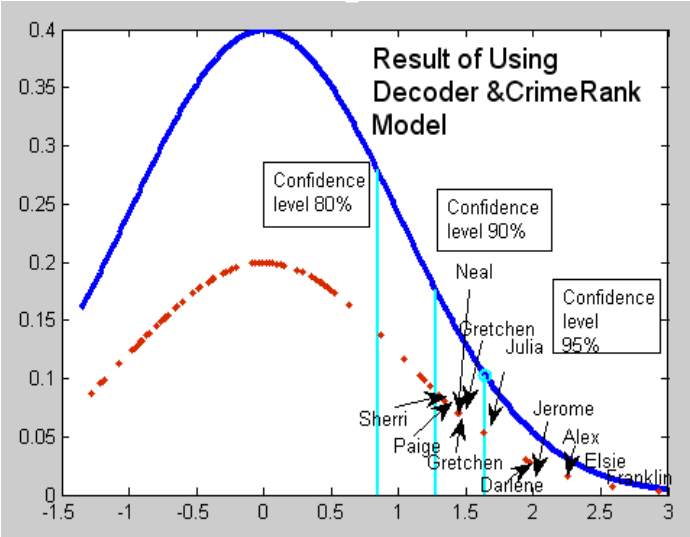

Figure 7

At 90% confidence level, Franklin, Elsie, Alex, Jerome, Darlene, Julia, Gretchen (No.4), Neal, Gretchen (No.32), Paige and Sherri are considered conspirators. Notice the result has shown the obvious difference with the previous conclusion, which displays the function of Decoder Model to handle with information in those 15 topics.

Details are shown in Appendix I.

Model Discussion

CrimeRank Model borrows idea from Google's famous PageRank Algorithm. It provides a general method to judge whether a person is conspirator in any initial state and given information. However, some topics must be known in advance whether they

are suspicious.

Decoder Model is design to solve the above-mentioned problem. It can catch sight of relative significance of atopic. Decoder Model could be widely used in many aspects such as text analysis, evidence verifying. We combine Decoder Model with CrimeRank Model together to automatically judge which topics are crucial and then seek out the conspirators.

Further improvement could be made to consummate Decoder Model. For instance, some particular words could be selected initially, which will be endowed with heavier weight to closely link them with the class of the crime. Take financial crime as example. We can pick up word “money”, “stock” at the very beginning to perfect the Decoder Model.

WorldNetwork: An Application in more general cases

Other than crime detection, our model could be applied in wider range of fields such as biological network, international relationship network, social network system, viral marketing, keywords detection, etc. For instance, when we use CrimeRank Model to deal with biological network problem, each cell could be treated as a person. An infected cell might be compared as a known conspirator. Likewise, virus spread among cells can be seen as topic communication among people. In this way we can duplicate the procedure in CrimeRank Model to detect whether a certain cell is infected.

Now we raise a concrete example in international relationship network to illustrate further application of CrimeRank Model.

During the period in Cold War, there are two military blocs: NATO and Warsaw Pact. We pick a period (1945-1976) to do the analysis to find out which countries are inclined to NATO and which to Warsaw Pact. 42 countries are selected to complete this model. Initiative members of Warsaw Pact can be treated as "conspirators" and those of NATO as "innocents". If the leader of country A once visited country B, it is regarded as a communication.

Figure 8a

Following the Flow Map Procedure in CrimeRank Model, we can finally get the result of which countries are inclined to NATO and which to Warsaw Pact.

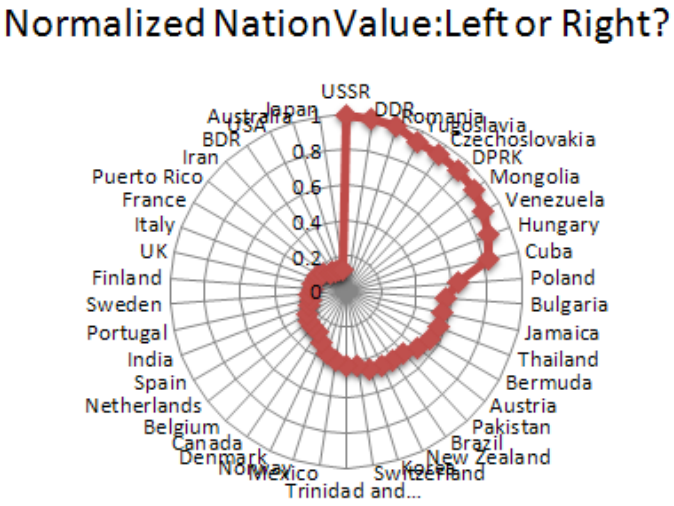

Figure 8b

Reference

[1]M.Chau, J.J. Xu, and H. Chen, "Extracting Meaningful Entities from Police Narrative Reports, Proc. Nat'l Conf. Digital Government Research, Digital Government Research Center, 2002, pp. 271-275.

[2]R.V. Hauck et al., "Using Coplink to Analyze Criminal-Justice Data," Computer, Mar. 2002, pp. 30-37.

[3]Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wan, Yi Qin and Michael Chau, "Crime Data Mining: A General Framework and Some Examples", Research Feature, 2004, pp.50-56.

[4]Marya. L. Doerfel, George A. Barnett, "A Semantic Network Analysis of the International Communication Association", Human Communication Research, Vol. 25 No. 4, June 1999, Human Communication Research, pp.589-603.

[5]Pedro Domingos, Matt Richardson, University of Washington, "Mining the Network Value of Customers", 2001.

[6]Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", Stanford University, 1998.

[7]Clarke, Ronald V. and John Eck, "Crime Analysis for Problem Solvers: in 60 Small Steps", U.S. Department of Justice, Office of Community Oriented Policing Services, 2005.

[8]R. I. M. Dunbar, University of Liverpool, "Gossip in Evolutionary Perspective", Review of General Psychology, 2004, Vol. 8, No. 2, pp.100 –110.

[9]G. Salton, A. Wong , C. S. Yang, "A vector space model for automatic indexing",

Communications of the ACM, v.18 n.11, p.613-620, Nov. 1975.

[10]Edgar Moyotl-Hernández and Héctor Jiménez-Salazar, "Enhancement of DTP Feature Selection Method for Text Categorization"

[11]Spärck Jones, Karen, "A statistical interpretation of term specificity and its application in retrieval", Journal of Documentation 28 (1), 1972, pp. 11–21.

Appendix I

Results of Requirement 1:

Normalized CrimeRank Value 1							
*Harvey	0.99998	Kristine	0.64437	Lois	0.368991	Bariol	0.249543
*Alex	0.999404	Dwight	0.585884	Gerry	0.349036	Marian	0.248433
$Dolores	0.999043	*Paul	0.576309	Claire	0.334255	Douglas	0.233168
*Yao	0.996918	William	0.560482	Kim	0.328856	Hark	0.220251
$Jerome	0.937861	Patrick	0.553223	Christina	0.328539	Chara	0.219732
#Darlene	0.915723	Beth	0.53274	Melia	0.324121	Cha	0.209765
$Gretchen	0.910603	Beth	0.530652	Mai	0.318477	Andra	0.209765
*Ulf	0.906284	Stephanie	0.510076	Louis	0.312564	Olina	0.202281
Karen	0.88254	Neal	0.506235	Malcolm	0.310089	Darol	0.202281
Neal	0.879582	Francis	0.50414	Quan	0.308238	#Ellin	0.19496
Franklin	0.848955	Priscilla	0.50414	Cory	0.307622	Seeni	0.17063
Fanti	0.824601	Crystal	0.481098	#Tran	0.306699	Dayi	0.169301
Elsie	0.820503	Wayne	0.472382	Marcia	0.303632	Vind	0.169301
Sheng	0.802547	Kristina	0.418457	Hazel	0.297843	Lao	0.169301
#Paige	0.779674	*Jean	0.417431	Cole	0.295723	Lars	0.169301
Donald	0.776287	Eric	0.415382	Han	0.287313	Le	0.169301
Gretchen	0.744504	*Elsie	0.40212	#Chris	0.273747	#Jia	0.169301
Julia	0.723832	Shelley	0.37994	Reni	0.27055	Carina	0.169301
Marion	0.699637	Sandy	0.378275	#Este	0.261928	#Gard	0.169301
Sherri	0.676417	Wesley	0.375947	Erica	0.256824	Phille	0.169301
Patricia	0.666926	Katherine	0.375947	Jerome	0.25289		
* indicates prior known conspirators;# indicates prior known non-conspirators;$ indicates senior managers							

Modified Normalized CrimeRank Value 1							
Franklin	0.99833	Christina	0.6819603	Francis	0.4173873	Andra	0.1970261
*Elsie	0.99508	*Ulf	0.6810744	William	0.4051563	Han	0.1926939
*Alex	0.98797	Patricia	0.6721593	Shelley	0.4013124	Darol	0.1831065
$Jerome	0.97605	Elsie	0.6655583	Lois	0.3888875	#Ellin	0.1813562
#Darlene	0.97422	Stephanie	0.6619359	Malcolm	0.3822431	$Jerome	0.1750995
Julia	0.948	Marion	0.6515916	Erica	0.3532569	Quan	0.1679408
$Gretchen	0.92606	Beth	0.6377265	Hazel	0.3511068	Reni	0.1646381
$Gretchen	0.92594	Eric	0.6151687	Marcia	0.3392166	Olina	0.1421471

Neal	0.92489	Crystal	0.6113685	Mai	0.318278	Cha	0.1183555
#Paige	0.91073	Priscilla	0.5970254	#Chris	0.3173929	Bariol	0.1157535
Sherri	0.90287	*Paul	0.5611221	#Este	0.3138629	Seeni	0.1125585
$Dolores	0.89178	Fanti	0.5604696	Marian	0.2993399	Dayi	0.0996827
Karen	0.88243	Sandy	0.5490262	Melia	0.271798	Vind	0.0996827
Beth	0.8783	Sheng	0.5395131	Gerry	0.2601244	Lao	0.0996827
Wesley	0.87578	Patrick	0.5240495	Cory	0.2595878	Lars	0.0996827
*Yao	0.85082	#Tran	0.5161377	Chara	0.2563802	Le	0.0996827
Kristine	0.80726	Kristina	0.5078896	Hark	0.2378331	#Jia	0.0996827
*Jean	0.73578	Neal	0.4930368	Cole	0.2324918	Carina	0.0996827
Dwight	0.70032	*Harvey	0.488087	Louis	0.2304747	#Gard	0.0996827
Donald	0.68901	Claire	0.4742398	Douglas	0.2210282	Phille	0.0996827
Wayne	0.68226	Katherine	0.4660111	Kim	0.2142118		

* indicates prior known conspirators, # indicates prior known non-conspirators,$ indicates senior managers

Result of Requirement 2:

Normalized CrimeRank Value 2							
*Harvey	0.999891	Francis	0.680388	Marcia	0.396749	Erica	0.215793
*Alex	0.999683	Marion	0.643217	Christina	0.389919	Marian	0.209699
$Dolores	0.996592	Crystal	0.610838	Sandy	0.38899	Bariol	0.205769
*Yao	0.992559	Sherri	0.606498	Lois	0.383123	Douglas	0.198042
#Darlene	0.960208	Kristine	0.594026	*Jean	0.36964	Chara	0.184625
$Jerome	0.946502	William	0.589951	Claire	0.363259	Hark	0.181627
*Ulf	0.88035	Eric	0.567536	*Elsie	0.346732	Andra	0.173025
Elsie	0.876438	Dwight	0.563088	#Tran	0.344351	Olina	0.166898
$Gretchen	0.865921	Paul	0.526621	Katherine	0.335479	Darol	0.166898
Melia	0.857003	*Chris	0.513112	Shelley	0.334597	#Ellin	0.164687
Julia	0.849398	Neal	0.501199	Cha	0.326406	Seeni	0.141082
Sheng	0.8364	Kristina	0.496368	Kim	0.302345	Dayi	0.140001
Karen	0.835202	Beth	0.487031	Gerry	0.28924	Vind	0.140001
Neal	0.833192	Patrick	0.484778	Louis	0.2802	Lao	0.140001
Quan	0.798899	Beth	0.469989	Mai	0.264917	Lars	0.140001
Franklin	0.79708	Stephanie	0.464533	Jerome	0.254192	Le	0.140001
Donald	0.757929	Priscilla	0.458122	Cory	0.254192	#Jia	0.140001
#Paige	0.75692	Han	0.426899	Hazel	0.253416	Carina	0.140001
Gretchen	0.750045	Wesley	0.425949	Cole	0.244205	#Gard	0.140001
Fanti	0.746182	Wayne	0.417422	Reni	0.226581	Phille	0.140001
Patricia	0.687281	Malcolm	0.416792	#Este	0.21603		

* indicates prior known conspirators, # indicates prior known non-conspirators,$ indicates senior managers

Modified Normalized CrimeRankValue 2							
Franklin	0.9980445	Crystal	0.6840462	Francis	0.4192327	Han	0.1993567
*Harvey	0.990098	Christina	0.6766852	Lois	0.3990704	Andra	0.1952395
Elsie	0.9778936	Patricia	0.6737211	Shelley	0.3984345	Darol	0.1814581
#Darlene	0.9702991	*Jean	0.6701497	William	0.3879808	Ellin	0.1788628
$Dolores	0.9574746	*Elsie	0.6490152	Malcolm	0.3779208	Jerome	0.1733203
$Jerome	0.9545625	Marion	0.6438091	Erica	0.3500873	Quan	0.1664449
Alex	0.9537701	Stephanie	0.6438091	Hazel	0.3473449	Reni	0.1631756
Julia	0.9458578	Beth	0.6264738	Marcia	0.3397693	Olina	0.1409138
$Gretchen	0.9239479	Eric	0.6072962	Mai	0.3145079	Cha	0.1173666
Gretchen	0.9239479	Priscilla	0.5772428	#Este	0.311007	Bariol	0.1147913
Neal	0.9196144	*Ulf	0.5753072	*Chris	0.2957522	Seeni	0.0990286
Beth	0.915097	Fanti	0.5522765	Marian	0.2883965	Dayi	0.0988852
Sherri	0.8836136	*Paul	0.541507	Melia	0.2647011	Vind	0.0988852
*Yao	0.8796976	Sandy	0.5411801	Gerry	0.2577374	Lao	0.0988852
Karen	0.8724512	Sheng	0.5352919	Cory	0.2572057	Lars	0.0988852
Wesley	0.8705496	Patrick	0.5336549	Chara	0.2542915	Le	0.0988852
#Paige	0.8436439	#Tran	0.5120032	Hark	0.2356527	#Jia	0.0988852
Kristine	0.8036358	Kristina	0.5031319	Cole	0.2303618	Carina	0.0988852
Dwight	0.6979746	Neal	0.4903161	Louis	0.2226743	#Gard	0.0988852
Donald	0.6919148	Claire	0.475542	Douglas	0.2134535	Phille	0.0988852
Wayne	0.6858021	Katherine	0.4608014	Kim	0.2120182		
* indicates prior known conspirators, # indicates prior known non-conspirators,$ indicates senior managers							

Appendix II

C++ code

CrimeRank Model

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
```

```
const double quan=0.22;    //Requirement 1: 104/354 Requirement 2: 133/325
const double conspiratorpercent=0.5;
const double e=0.00000001;
const double d=0.8;

int main()
{
                                        FILE *stream,*stream2;
                                                int
i,j,k,nodenum,linknum,effecttopicnum,effecttopic[10],conspiratornum,conspirator[10],n
conspiratornum,nconspirator[10]
                        ,startnode=0,endnode=0,topicnum,topic,flag,source;
                                              double
node[100][20],link[100][100],sort[100][2],conspiratorvalue,othersvalue,pr,value,ec,sum
=0,temp0,temp1;

                                        for(i=0;i<100;i++)
                                                for (j=0;j<20;j++)
                                                    node[i][j]=0;

                                        for(i=0;i<100;i++)
                                                for (j=0;j<100;j++)
                                                    link[i][j]=0;

                    if((stream=fopen("data.txt","rt"))==NULL)
                                                printf("Cannot open the file!");
                else if ((stream2=fopen("data.dat","wb"))==NULL)
                                                    printf("Cannot open the file!");
                                            else
                                                {
                                                  while ( feof(stream)==0 )
                                                  {

fscanf(stream,"%d",&nodenum);

fscanf(stream,"%d",&linknum);

fscanf(stream,"%d",&effecttopicnum);

                                                                for
(i=0;i<effecttopicnum;i++)

fscanf(stream,"%d",&effecttopic[i]);
```

```
fscanf(stream,"%d",&conspiratornum);

                                                    for
(i=0;i<conspiratornum;i++)

fscanf(stream,"%d",&conspirator[i]);


conspiratorvalue=conspiratorpercent*1/conspiratornum;


fscanf(stream,"%d",&nconspiratornum);

                                                    for
(i=0;i<nconspiratornum;i++)

fscanf(stream,"%d",&nconspirator[i]);

                                                //      nconspiratorvalue=0;

othersvalue=(1-conspiratorpercent)*1/(nodenum-conspiratornum-nconspiratornum);

                                                    for (i=0;i<linknum;i++)
                                                    {

fscanf(stream,"%d",&startnode);

fscanf(stream,"%d",&endnode);

fscanf(stream,"%d",&topicnum);
                                                        for (j=0;j<topicnum;j++)
                                                        {

fscanf(stream,"%d",&topic);
                                                            value=quan;
                                                            for

(k=0;k<effecttopicnum;k++)
                                                            {
                                                                if

(topic==effecttopic[k])
                                                                    value=1;
                                                            }
```

```
                                                    node[startnode][1]
++;

link[startnode][endnode] +=value;
                                                        }
                                                    }
                                                }
                                            }

                                    for (i=0;i<nodenum;i++)
                                            {
                                                node[i][0]=othersvalue;
                                                for (j=0;j<conspiratornum;j++)
                                                {
                                                    if (i==conspirator[j])
                                                        node[i][0]=conspiratorvalue;
                                                }
                                                for (j=0;j<nconspiratornum;j++)
                                                {
                                                    if (i==nconspirator[j])
                                                        node[i][0]=0;
                                                }

                                                k=3;
                                                for (j=0;j<nodenum;j++)
                                                {
                                                //    node[i][1] += link[i][j];
                                                    if (link[j][i]!=0)
                                                    {
                                                        node[i][2]++;
                                                        node[i][k]=j;
                                                        k++;
                                                    }
                                                }
                                            }

                                        flag=1;
                                    while (flag==1)
                                            {
                                                flag=0;
                                                for (i=0;i<nodenum;i++)
                                                {
                                                    pr=(1-d)/nodenum;
                                                    for (j=1;j<=node[i][2];j++)
```

```
                                        {
                                            source=node[i][2+j];
                                            pr +=
d*node[source][0]*link[source][i]/node[source][1];
                                        }
                                        if (pr>node[i][0])
                                            ec=pr-node[i][0];
                                        else
                                            ec=node[i][0]-pr;
                                        if ( ec>e )
                                              flag=1;
                                        node[i][0]=pr;
                                    }
                                }

                            for (i=0;i<nodenum;i++)
                                        sum += node[i][0];
                              printf("%lf\n",sum);

                            for (i=0;i<nodenum;i++)
                                    {
                                        fprintf(stream2,"%d\t",i);
                                        fprintf(stream2,"%lf\n",node[i][0]);
                                    //  fprintf(stream2,"\n");
                                    }

                              fprintf(stream2,"\n");

                            for (i=0;i<nodenum;i++)
                                    {
                                        sort[i][0]=i;
                                        sort[i][1]=node[i][0];
                                    }

                            for (i=1;i<nodenum;i++)
                                    {
                                        for (j=0;j<nodenum-1;j++)
                                        {
                                            if (sort[j][1]<sort[j+1][1])
                                            {
                                                temp0=sort[j][0];
                                                temp1=sort[j][1];
                                                sort[j][0]=sort[j+1][0];
                                                sort[j][1]=sort[j+1][1];
```

```
                                        sort[j+1][0]=temp0;
                                        sort[j+1][1]=temp1;
                                    }
                                }
                            }

                    for (i=0;i<nodenum;i++)
                            {
                                fprintf(stream2,"%f\t",sort[i][0]);
                                fprintf(stream2,"%lf\n",sort[i][1]);
                            //  fprintf(stream2,"\n");
                            }

                    fclose(stream);
                    fclose(stream2);
                        return;

}
```

CrimeSource Model

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>

const double quan=0.22;
const int infinite=10000;
const double etotal=10;
const double itotal=10;
//const double conspiratorpercent=0.5;
//const double e=0.001;
//const double d=0.8;

int main()
{
                        FILE *stream,*stream2;
                                int
i,j,k,l,nodenum,linknum,effecttopicnum,effecttopic[10],conspiratornum,conspirator[10],
nconspiratornum,nconspirator[10]
                ,startnode=0,endnode=0,topicnum,topic,flag,source;
                                double
node[100][3],elink[100][100],ilink[100][100],sort[100][2],conspiratorvalue,othersvalue,
```

```
pr,value,ec,sum=0,temp0,temp1;

                    for(i=0;i<100;i++)
                            for (j=0;j<20;j++)
                                node[i][j]=0;

                    for(i=0;i<100;i++)
                            for (j=0;j<100;j++)
                            {
                                if (i==j)
                                    elink[i][j]=infinite;
                                else elink[i][j]=1/etotal;
                            }

                    for(i=0;i<100;i++)
                            for (j=0;j<100;j++)
                            {
                                if (i==j)
                                    ilink[i][j]=infinite;
                                else ilink[i][j]=1/itotal;
                            }

            if((stream=fopen("data.txt","rt"))==NULL)
                            printf("Cannot open the file!");
        else if ((stream2=fopen("data.dat","wb"))==NULL)
                                printf("Cannot open the file!");
                        else
                             {
                               while ( feof(stream)==0 )
                               {

fscanf(stream,"%d",&nodenum);

fscanf(stream,"%d",&linknum);

fscanf(stream,"%d",&effecttopicnum);

                                                for
(i=0;i<effecttopicnum;i++)

fscanf(stream,"%d",&effecttopic[i]);


fscanf(stream,"%d",&conspiratornum);
```

```
                                                    for
(i=0;i<conspiratornum;i++)

fscanf(stream,"%d",&conspirator[i]);

                                                //
conspiratorvalue=conspiratorpercent*1/conspiratornum;


fscanf(stream,"%d",&nconspiratornum);

                                                    for
(i=0;i<nconspiratornum;i++)

fscanf(stream,"%d",&nconspirator[i]);

                                                //      nconspiratorvalue=0;
                                                //
othersvalue=(1-conspiratorpercent)*1/(nodenum-conspiratornum-nconspiratornum);

                                                    for (i=0;i<linknum;i++)
                                                    {

fscanf(stream,"%d",&startnode);

fscanf(stream,"%d",&endnode);

fscanf(stream,"%d",&topicnum);
                                                        for (j=0;j<topicnum;j++)
                                                        {


fscanf(stream,"%d",&topic);
                                                            value=quan;
                                                            for

(k=0;k<effecttopicnum;k++)
                                                            {
                                                                if

(topic==effecttopic[k])
                                                                    value=1;
                                                            }


                                                        //      node[startnode][1]
+=value;
```

```
ilink[startnode][endnode] += value;

ilink[endnode][startnode] += value;
/*
                                                    flag=0;
                                                    for
(k=0;k<conspiratornum;k++)
                                                    {
                                                        l=conspirator[k];
                                                        if (startnode==l)
                                                            flag=1;
                                                        if (endnode==l)
                                                            flag=1;
                                                    }
                                                    if (flag==0)
                                                    {
*/

elink[startnode][endnode] += value;

elink[endnode][startnode] += value;
                                                //              }
                                                            }
                                                        }
                                                    }
                                                }

                                    for (i=0;i<nodenum;i++)
                                            {
                                                for (j=0;j<nodenum;j++)
                                                {
                                                    elink[i][j]=1/elink[i][j];
                                                    ilink[i][j]=1/ilink[i][j];
                                                }
                                            }

    for (k=0;k<nodenum;k++)
                                            {
                                                for (i=0;i<nodenum;i++)
                                                {
                                                    for (j=0;j<nodenum;j++)
                                                    {
                                                        if
```

```
(elink[i][k]+elink[j][k]<elink[i][j])
                                                    {

                                elink[i][j]=elink[i][k]+elink[j][k];
                                                        elink[j][i]=elink[i][j];
                                                    }
                                                    if
(ilink[i][k]+ilink[j][k]<ilink[i][j])
                                                    {

                                 ilink[i][j]=ilink[i][k]+ilink[j][k];
                                                        ilink[j][i]=ilink[i][j];
                                                    }
                                                }
                                            }
                                        }

                                    for (i=0;i<nodenum;i++)
                                            {
                                                flag=0;
                                                for (j=0;j<conspiratornum;j++)
                                                {
                                                    if (i==conspirator[j])
                                                    {
                                                        node[i][2]=etotal;
                                                        flag=1;
                                                    }
                                                }
                                                for (j=0;j<nconspiratornum;j++)
                                                {
                                                    if (i==nconspirator[j])
                                                    {
                                                        node[i][2]=-itotal;
                                                        flag=1;
                                                    }
                                                }
                                                if (flag==0)
                                                {
                                                    for (j=0;j<conspiratornum;j++)
                                                    {
                                                     k=conspirator[j];
                                                     node[i][0]                    =
node[i][0]+etotal/elink[i][k];
                                                    }
```

```
                                        for (j=0;j<nconspiratornum;j++)
                                        {
                                            k=nconspirator[j];
                                            node[i][1]                    =
node[i][1]+itotal/ilink[i][k];
                                        }
                                        node[i][2]=node[i][0]-node[i][1];
                                    }
                                }

/*
                            for (i=1;i<=nodenum;i++)
                                {
                                    node[i][0]=othersvalue;
                                    for (j=0;j<conspiratornum;j++)
                                    {
                                        if (i==conspirator[j])
                                            node[i][0]=conspiratorvalue;
                                    }
                                    for (j=0;j<nconspiratornum;j++)
                                    {
                                        if (i==nconspirator[j])
                                            node[i][0]=0;
                                    }

                                    k=3;
                                    for (j=1;j<=nodenum;j++)
                                    {
                                    //    node[i][1] += link[i][j];
                                        if (link[j][i]!=0)
                                        {
                                            node[i][2]++;
                                            node[i][k]=j;
                                            k++;
                                        }
                                    }
                                }

                                flag=1;
                            while (flag==1)
                                {
                                    flag=0;
                                    for (i=1;i<=nodenum;i++)
                                    {
```

```
                                        pr=(1-d)/nodenum;
                                        for (j=1;j<=node[i][2];j++)
                                        {
                                            source=node[i][2+j];
                                            pr                              +=
d*node[source][0]*link[source][i]/node[source][1];
                                        }
                                        if (pr>node[i][0])
                                            ec=pr-node[i][0];
                                        else
                                            ec=node[i][0]-pr;
                                        if ( ec>e )
                                              flag=1;
                                        node[i][0]=pr;
                                    }
                                }
*/
                            for (i=0;i<nodenum;i++)
                                        sum += node[i][2];
                              printf("%lf\n",sum);

                            for (i=0;i<nodenum;i++)
                                    {
                                        fprintf(stream2,"%d\t",i);
                                        fprintf(stream2,"%lf\n",node[i][2]);
                                    //  fprintf(stream2,"\n");
                                    }

                              fprintf(stream2,"\n");

                            for (i=0;i<nodenum;i++)
                                    {
                                        sort[i][0]=i;
                                        sort[i][1]=node[i][2];
                                    }

                            for (i=1;i<nodenum;i++)
                                    {
                                        for (j=0;j<nodenum-1;j++)
                                        {
                                            if (sort[j][1]<sort[j+1][1])
                                            {
                                                temp0=sort[j][0];
                                                temp1=sort[j][1];
```

```
                                        sort[j][0]=sort[j+1][0];
                                        sort[j][1]=sort[j+1][1];
                                        sort[j+1][0]=temp0;
                                        sort[j+1][1]=temp1;
                                    }
                                }
                            }

                        for (i=0;i<nodenum;i++)
                            {
                                fprintf(stream2,"%f\t",sort[i][0]);
                                fprintf(stream2,"%lf\n",sort[i][1]);
                            //  fprintf(stream2,"\n");
                            }

                        fclose(stream);
                        fclose(stream2);
                            return;

}
```

Decoder Model

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>

const double e=0.000000001;
const double d=0.8;
/*
const double quan=104/353;
const double conspiratorpercent=0.5;

*/
int main()
{
                        FILE *stream,*stream2;
                                int
topic[20][200],i,j,k,l,m,nodenum,topicnum,nod,nodetopic,top,topicnode,currentnode,fla
g,source;
      double node[200][200],pr,ec,sum=0,sort[200][2],topicvalue[20],temp0,temp1;

                                //
,linknum,effecttopicnum,effecttopic[10],conspiratornum,conspirator[10],nconspiratornu
m,nconspirator[10]
//                          ,startnode=0,endnode=0;
```

```
//,conspiratorvalue,othersvalue,,value,;

for(i=0;i<200;i++)
    for (j=0;j<200;j++)
        node[i][j]=0;

for(i=0;i<20;i++)
    for (j=0;j<200;j++)
        topic[i][j]=0;

for (i=0;i<20;i++)
    topicvalue[i]=0;

if((stream=fopen("data.txt","rt"))==NULL)
    printf("Cannot open the file!");
else if ((stream2=fopen("data.dat","wb"))==NULL)
    printf("Cannot open the file!");
else
{
    while ( feof(stream)==0 )
    {

fscanf(stream,"%d",&nodenum);

fscanf(stream,"%d",&topicnum);

        for (i=0;i<nodenum;i++)
        {

fscanf(stream,"%d",&nod);

fscanf(stream,"%d",&nodetopic);
            //
fscanf(stream,"%d",&topicnum);
            for (j=0;j<nodetopic;j++)
            {

fscanf(stream,"%d",&top);
                k=topic[top][0];
                topic[top][1+k]=nod;
                topic[top][0]++;
            }
        }
        /*
```

```
                                                    value=quan;
                                                        for
(k=0;k<effecttopicnum;k++)
                                                        {
                                                            if
(topic==effecttopic[k])
                                                                value=1;
                                                        }

                                                        node[startnode][1]
++;

link[startnode][endnode] +=value;
                                                        */
                                                    }
                                                }


      for (i=1;i<=topicnum;i++)
                                            {
                                                topicnode=topic[i][0];
                                                for (j=1;j<topicnode;j++)
                                                {
                                                    currentnode=topic[i][j];

                                                    for (k=1;k<=topicnode-j;k++)
                                                    {
                                                        flag=0;
                                                        for
(l=1;l<=node[currentnode][1];l++)
                                                        {
                                                            if
(topic[i][j+k]==node[currentnode][1+l])
                                                                flag=1;
                                                        }
                                                        if (flag==0)
                                                        {
                                                            l=node[currentnode][1];

node[currentnode][2+l]=topic[i][j+k];
                                                            node[currentnode][1]++;
                                                            m=topic[i][j+k];
                                                            l=node[m][1];
```

```
                    node[m][2+l]=currentnode;
                                        node[m][1]++;
                                    }
                                }
                            }
                        }

                    for (i=0;i<nodenum;i++)
                        {
                            node[i][0]=1;
                        }

                        flag=1;
                    while (flag==1)
                        {
                            flag=0;
                            for (i=0;i<nodenum;i++)
                            {
                                pr=(1-d)/nodenum;
                                for (j=1;j<=node[i][1];j++)
                                {
                                    source=node[i][1+j];
                                    pr                          +=
d*node[source][0]/node[source][1];
                                }
                                if (pr>node[i][0])
                                    ec=pr-node[i][0];
                                else
                                    ec=node[i][0]-pr;
                                if ( ec>e )
                                      flag=1;
                                node[i][0]=pr;
                            }
                        }

                    for (i=1;i<=topicnum;i++)
                        {
                            for (j=1;j<=topic[i][0];j++)
                            {
                                k= topic[i][j];
                                topicvalue[i] += node[k][0];
                            }
                            topicvalue[i] /= topic[i][0];
                            fprintf(stream2,"%d\t",i);
```

```
                    fprintf(stream2,"%lf\n",topicvalue[i]);
                }

 fprintf(stream2,"\n\n");

for (i=0;i<nodenum;i++)
                    sum += node[i][0];
   printf("%lf\n",sum);

for (i=0;i<nodenum;i++)
                {
                    fprintf(stream2,"%d\t",i);
                    fprintf(stream2,"%lf\n",node[i][0]);
                //  fprintf(stream2,"\n");
                }

  fprintf(stream2,"\n");

for (i=0;i<nodenum;i++)
                {
                    sort[i][0]=i;
                    sort[i][1]=node[i][0];
                }

for (i=1;i<nodenum;i++)
                {
                    for (j=0;j<nodenum-1;j++)
                    {
                        if (sort[j][1]<sort[j+1][1])
                        {
                            temp0=sort[j][0];
                            temp1=sort[j][1];
                            sort[j][0]=sort[j+1][0];
                            sort[j][1]=sort[j+1][1];
                            sort[j+1][0]=temp0;
                            sort[j+1][1]=temp1;
                        }
                    }
                }

for (i=0;i<nodenum;i++)
                {
                    fprintf(stream2,"%f\t",sort[i][0]);
                    fprintf(stream2,"%lf\n",sort[i][1]);
```

```
                                        //    fprintf(stream2,"\n");
                                        }

                                    fclose(stream);
                                   fclose(stream2);
                                        return;

}
```

Modified CrimeRank Model

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>

//const double quan=104/353;
const double conspiratorpercent=0.5;
const double e=0.00000001;
const double d=0.8;

int main()
{
                                    FILE *stream,*stream2;
                                            int
i,j,k,nodenum,linknum,topictotalnum,conspiratornum,conspirator[10],nconspiratornum,
nconspirator[10]
                            ,startnode=0,endnode=0,topicnum,topic,flag,source;
                                            double
node[100][20],link[100][100],sort[100][2],topicvalue[20],conspiratorvalue,othersvalue,
pr,value,ec,sum=0,temp0,temp1;

                                //effecttopicnum,effecttopic[10],

                                for(i=0;i<100;i++)
                                        for (j=0;j<20;j++)
                                            node[i][j]=0;

                                for(i=0;i<100;i++)
                                        for (j=0;j<100;j++)
                                            link[i][j]=0;
                                for (i=0;i<20;i++)
                                        topicvalue[i]=0;
```

```
                if((stream=fopen("data.txt","rt"))==NULL)
                                printf("Cannot open the file!");
            else if ((stream2=fopen("data.dat","wb"))==NULL)
                                printf("Cannot open the file!");
                        else
                            {
                              while ( feof(stream)==0 )
                              {

fscanf(stream,"%d",&nodenum);

fscanf(stream,"%d",&linknum);

fscanf(stream,"%d",&topictotalnum);

                                        for
(i=1;i<=topictotalnum;i++)
                                        {
                                            fscanf(stream,"%d",&j);

fscanf(stream,"%lf",&topicvalue[j]);
                                        }


fscanf(stream,"%d",&conspiratornum);

                                        for
(i=0;i<conspiratornum;i++)

fscanf(stream,"%d",&conspirator[i]);


conspiratorvalue=conspiratorpercent*1/conspiratornum;


fscanf(stream,"%d",&nconspiratornum);

                                        for
(i=0;i<nconspiratornum;i++)

fscanf(stream,"%d",&nconspirator[i]);

                                //      nconspiratorvalue=0;
```

```
othersvalue=(1-conspiratorpercent)*1/(nodenum-conspiratornum-nconspiratornum);

                                                    for (i=0;i<linknum;i++)
                                                    {

fscanf(stream,"%d",&startnode);

fscanf(stream,"%d",&endnode);

fscanf(stream,"%d",&topicnum);
                                                        for (j=0;j<topicnum;j++)
                                                        {

fscanf(stream,"%d",&topic);
                                                    //      value=quan;
                                                    //      for
(k=1;k<=topictotalnum;k++)
                                                    //      {
                                                    //          if (topic==k)
                                                    //              value=1;
                                                    //      }

                                                            node[startnode][1]
++;

link[startnode][endnode] += topicvalue[topic];
                                                        }
                                                    }
                                                }
                                            }

                                    for (i=0;i<nodenum;i++)
                                        {
                                            node[i][0]=othersvalue;
                                            for (j=0;j<conspiratornum;j++)
                                            {
                                                if (i==conspirator[j])
                                                    node[i][0]=conspiratorvalue;
                                            }
                                            for (j=0;j<nconspiratornum;j++)
                                            {
                                                if (i==nconspirator[j])
                                                    node[i][0]=0;
                                            }
```

```
                    k=3;
                    for (j=0;j<nodenum;j++)
                    {
                    //    node[i][1] += link[i][j];
                        if (link[j][i]!=0)
                        {
                            node[i][2]++;
                            node[i][k]=j;
                            k++;
                        }
                    }
                }

            flag=1;
        while (flag==1)
            {
                flag=0;
                for (i=0;i<nodenum;i++)
                {
                    pr=(1-d)/nodenum;
                    for (j=1;j<=node[i][2];j++)
                    {
                        source=node[i][2+j];
                        pr                              +=
d*node[source][0]*link[source][i]/node[source][1];
                    }
                    if (pr>node[i][0])
                        ec=pr-node[i][0];
                    else
                        ec=node[i][0]-pr;
                    if ( ec>e )
                          flag=1;
                    node[i][0]=pr;
                }
            }

        for (i=0;i<nodenum;i++)
                    sum += node[i][0];
      printf("%lf\n",sum);

        for (i=0;i<nodenum;i++)
            {
                fprintf(stream2,"%d\t",i);
```

```
                    fprintf(stream2,"%lf\n",node[i][0]);
                //  fprintf(stream2,"\n");
                }

  fprintf(stream2,"\n");

for (i=0;i<nodenum;i++)
                {
                    sort[i][0]=i;
                    sort[i][1]=node[i][0];
                }

for (i=1;i<nodenum;i++)
                {
                    for (j=0;j<nodenum-1;j++)
                    {
                        if (sort[j][1]<sort[j+1][1])
                        {
                            temp0=sort[j][0];
                            temp1=sort[j][1];
                            sort[j][0]=sort[j+1][0];
                            sort[j][1]=sort[j+1][1];
                            sort[j+1][0]=temp0;
                            sort[j+1][1]=temp1;
                        }
                    }
                }

for (i=0;i<nodenum;i++)
                {
                    fprintf(stream2,"%f\t",sort[i][0]);
                    fprintf(stream2,"%lf\n",sort[i][1]);
                //  fprintf(stream2,"\n");
                }

     fclose(stream);
    fclose(stream2);
          return;

}
```

WorldNet Model

```
#include<iostream.h>
#include<stdlib.h>
```

```
#include<stdio.h>
#include<math.h>

const double quan=0.22;    //Requirement 1: 104/354 Requirement 2: 133/325
const double conspiratorpercent=0.5;
const double e=0.00000001;
const double d=0.8;

int main()
{
                                        FILE *stream,*stream2;
                                                int
i,j,k,nodenum,linknum,effecttopicnum,effecttopic[10],conspiratornum,conspirator[20],n
conspiratornum,nconspirator[20]
                            ,startnode=0,endnode=0,topicnum,topic,flag,source;
                                              double
node[100][100],link[100][100],sort[100][2],conspiratorvalue,othersvalue,pr,value,ec,su
m=0,temp0,temp1;

                                        for(i=0;i<100;i++)
                                                for (j=0;j<100;j++)
                                                    node[i][j]=0;

                                        for(i=0;i<100;i++)
                                                for (j=0;j<100;j++)
                                                    link[i][j]=0;

                        if((stream=fopen("data.txt","rt"))==NULL)
                                                printf("Cannot open the file!");
                    else if ((stream2=fopen("data.dat","wb"))==NULL)
                                                    printf("Cannot open the file!");
                                            else
                                                {
                                                  while ( feof(stream)==0 )
                                                  {

fscanf(stream,"%d",&nodenum);

fscanf(stream,"%d",&linknum);

fscanf(stream,"%d",&effecttopicnum);

                                                                    for
(i=0;i<effecttopicnum;i++)
```

```
fscanf(stream,"%d",&effecttopic[i]);

fscanf(stream,"%d",&conspiratornum);
                                                    for
(i=0;i<conspiratornum;i++)
fscanf(stream,"%d",&conspirator[i]);

conspiratorvalue=conspiratorpercent*1/conspiratornum;

fscanf(stream,"%d",&nconspiratornum);
                                                    for
(i=0;i<nconspiratornum;i++)
fscanf(stream,"%d",&nconspirator[i]);
                                              //    nconspiratorvalue=0;
othersvalue=(1-conspiratorpercent)*1/(nodenum-conspiratornum-nconspiratornum);
                                                    for (i=0;i<linknum;i++)
                                                    {
fscanf(stream,"%d",&startnode);
fscanf(stream,"%d",&endnode);
fscanf(stream,"%d",&topicnum);
                                                        for (j=0;j<topicnum;j++)
                                                        {
fscanf(stream,"%d",&topic);
                                                            value=quan;
                                                            for
(k=0;k<effecttopicnum;k++)
                                                            {
                                                                if
(topic==effecttopic[k])
```

```
                                                    value=1;
                                                }

                                                node[startnode][1]
++;

link[startnode][endnode] +=value;
                                            }
                                        }
                                    }
                                }

                        for (i=0;i<nodenum;i++)
                            {
                                node[i][0]=othersvalue;
                                for (j=0;j<conspiratornum;j++)
                                {
                                    if (i==conspirator[j])
                                        node[i][0]=conspiratorvalue;
                                }
                                for (j=0;j<nconspiratornum;j++)
                                {
                                    if (i==nconspirator[j])
                                        node[i][0]=0;
                                }

                                k=3;
                                for (j=0;j<nodenum;j++)
                                {
                                //    node[i][1] += link[i][j];
                                    if (link[j][i]!=0)
                                    {
                                        node[i][2]++;
                                        node[i][k]=j;
                                        k++;
                                    }
                                }
                            }

                        flag=1;
                    while (flag==1)
                            {
                                flag=0;
                                for (i=0;i<nodenum;i++)
```

```
                                        {
                                            pr=(1-d)/nodenum;
                                            for (j=1;j<=node[i][2];j++)
                                            {
                                                source=node[i][2+j];
                                                pr                                  +=
d*node[source][0]*link[source][i]/node[source][1];
                                            }
                                            if (pr>node[i][0])
                                                ec=pr-node[i][0];
                                            else
                                                ec=node[i][0]-pr;
                                            if ( ec>e )
                                                flag=1;
                                            node[i][0]=pr;
                                        }
                                    }

                                for (i=0;i<nodenum;i++)
                                            sum += node[i][0];
                                  printf("%lf\n",sum);

                                for (i=0;i<nodenum;i++)
                                        {
                                            fprintf(stream2,"%d\t",i);
                                            fprintf(stream2,"%lf\n",node[i][0]);
                                        //  fprintf(stream2,"\n");
                                        }

                                  fprintf(stream2,"\n");

                                for (i=0;i<nodenum;i++)
                                        {
                                            sort[i][0]=i;
                                            sort[i][1]=node[i][0];
                                        }

                                for (i=1;i<nodenum;i++)
                                        {
                                            for (j=0;j<nodenum-1;j++)
                                            {
                                                if (sort[j][1]<sort[j+1][1])
                                                {
                                                    temp0=sort[j][0];
```

```
                                        temp1=sort[j][1];
                                        sort[j][0]=sort[j+1][0];
                                        sort[j][1]=sort[j+1][1];
                                        sort[j+1][0]=temp0;
                                        sort[j+1][1]=temp1;
                                    }
                                }
                            }

                    for (i=0;i<nodenum;i++)
                            {
                                fprintf(stream2,"%f\t",sort[i][0]);
                                fprintf(stream2,"%lf\n",sort[i][1]);
                            //  fprintf(stream2,"\n");
                            }

                     fclose(stream);
                    fclose(stream2);
                        return;

}
```