DEPARTMENT OF COMPUTER SCIENCE, RENSSELAER POLYTECHNIC INSTITUTE

Read Me

Lingxun Hu

October 2, 2012

1 The Matching Problem

In a social network, each node is able to send or receive messages. Each node can send messages to any other nodes in the network except to itself. The only information we can acquire is whether a node sent or received a message and the time it sent or received that message.

Our goal is to find as many as possible valid matchings between the sent messages and received messages, based on the information we can get. By analyzing the matchings, we can conclude which nodes are communicating with each other and find the relationship in the social network.

We use two different algorithms to do this work: dynamic programming method and chain algorithm. You could find these two programs in "Dynamic Matching.cpp" and "Chain Algorithm.cpp". More information about the solutions, the rules and the algorithms will be provided in the following text.

2 Rules

2.1 Step Function Rule

We use a step function to determine whether a sent message and a received message can be matched. We calculate a time interval : $T_{int} = T_{receive} - T_{send}$. Apparently, a message might not be received before it is sent. We also define T_{max} and T_{min}.

If $T_{int} > T_{max}$, which means the time interval is too large, then the sent message and the received message should not be matched. If $T_{int} < T_{min}$, which means the time interval is too small, then the sent message and received message should not be matched, either. Only when $T_{min} < T_{int} < T_{max}$, we regard it as a valid matching.

We define T_{max} and T_{min} at the beginning of each program. Of course you may define your own T_{max} and T_{min} according to the specific situations you are using our program.

2.2 Causality Rule

Causality rule in this matching problem means that a message sent earlier tends to be received earlier.

Suppose we have two sent messages S_1 and S_2 (S_1 is earlier than S_2) and two received messages R_1 and R_2 (R_1 is earlier than R_2). Simply using the step function, it is ok to match $S_1 \rightarrow R_2$ and $S_2 \rightarrow R_1$. However, it is forbidden in causality rule.

2.3 No-Loop Rule

As mentions in Section 1, a node cannot send a message to itself. We may encounter a possible matching, whose sent message and received message are from a same node. Though this possible matching obeys step function and causality rule, it is still invalid.

3 Algorithms

3.1 Dynamic Programming

We use dynamic programming algorithm to find all valid matchings in the program named "Dynamic Matching.cpp". This program finds the maximum weight matching from the input. You can find more about dynamic programming method online or from a textbook.

In order to find the maximum weight, we need a weight function. We define the weight function as a linearly descending function. Thus a matching will have more weight, if the time interval is smaller. (Of course the matching has to obey the three rules mentioned above.) Certainly, you could define your own weight function. Please go to "Match_Weight" function in "Dynamic Matching.cpp", if you would like to.

I set the program to output the running time of the program. It you change the size of your input data, you will find that the running time is quadratic with respect to the size of your input data. It is the best a dynamic programming method can do.

3.2 Chain Algorithm

The chain algorithm is mentioned on page 7 of the paper "Extracting Hidden Groups and their Structure from Streaming Interaction Data" by Mark K. Goldberg, Mykola Hayvanovych, Malik Magdon-Ismail, and William A. Wallace.

Our chain algorithm is a little bit different from the one in the paper, because we add a "no loop" constraint. Therefore, every time the program encounters a loop it will get a random number 0 or 1. 0 means to move the pointer of the sender vector forward, and 0 means to move the pointer of the receiver vector forward.

I set the program to output the running time of the program. It you change the size of your input data, you will find that the running time is linear with respect to the size of your input data, which is pretty efficient.

4 Input & Output

4.1 Input Format

Input data has two parts: "send.txt" and "receive.txt".

"send.txt" contains the information of all the 5000 sent messages. Each row represents a sent message. The integer number is the number ID of a node. It indicates which node sent this message. The double number indicates the time when the message was sent.

"receive.txt" has the same format as "send.txt". The integer indicates which node received this message. The double indicates the time when the message was received.

4.2 Output Format

"Dynamic matching.cpp" has output named "dynamic_matching.txt". Each row represent a valid matching. The first column is the number ID of the node which sent the message. The second column is the number ID of the node which received the message. The third column is the time when the message was sent.

"Chain algorithm.cpp" has output named "chain algorithm.txt", which has the same format as "dynamic_matching.txt".

4.3 Something About Input & Output

Because I do not have real network data available, I generated the input data to test my programs. If you could use these program to work on real data, that's pretty awesome!

I also set both programs to output the number of valid matchings it found. For dynamic programming method it is 4950 out of 5000 messages. And for chain algorithm it is 4965 out of 5000 messages. Pretty good result!

The output of my program can be used as the input of the program "Extracting Hidden Groups and their Structure from Streaming Interaction Data", which will help you find the hidden groups and their structure by analyzing my output data. You can also find that program on LFD-Lab website.

5 Something Else

If you could test the program on real data or have found a bug or have a problem, please contact the programmer:

Lingxun Hu
hul5@rpi.edu