Multi-resolution Overlay Measurement

-- Simulation Code Description

**1. Introduction:**

In this document, we introduce the codes of our simulation to do multi-resolution overlay measurement. In multi-resolution overlay measurement, we select parts of overlay links or overlay nodes to send probing packets among them to infer the underlay link conditions. Based on the overlay link measurement results, two methods: Maximum Likelihood Estimation and Bayesian Inference Model could be utilized to infer the underlay link situations.

**2. Code Overview:**

Basically, the simulation for the Multi-resolution overlay measurement consists of two parts: MATLAB simulation and C Code inference. In MATLAB simulation, after reading network configuration file, we will form network topology, randomly generate link failure and then use our proposed algorithms to infer underlay conditions. For the C Code inference, we parse the experiment results got from Emulab, use the proposed algorithms to infer the congestion underlay. For those two types of experiments, we will calculate the inference accuracy to test performances. We will describe each of the simulations in more details in the following sections.

**3. MATLAB Simulation:**

In MATLAB simulation, we test the performance of our overlay measurement by using two inference methods: Maximum Likelihood Estimation and Bayesian Inference Model respectively.

**3.1 Simulation Steps:**

The simulation runs as follows:

1) We will use one tool GT-ITM to generate underlay network. To make underlay network denser, we will randomly connect some of the nodes in the network to add more links.

2) We will randomly select subset of underlay nodes as overlay nodes. We will use shortest path algorithm to calculate the underlay paths among those overlay node pairs. It is possible that some underlay links are not traversed by any overlay pair, which means that overlay probing among the selected overlay nodes will never cover those parts of the underlay links. In our simulation, we will exclude those underlay links. After these two steps, we formulate the network test topology.

3) We assume that each underlay link e fails or becomes congested according to a fixed probability α,different links have different probabilities, following one of the three distributions: a) α is randomly chosen between 0 and 0.3; b) α follows a Normal Distribution with mean 0.05 and standard deviation of 0.1 - N(0.05, 0.12); c) α takes one of the three values: 70% links have probability of 0.01, 20% links have probability of 0.05, 10% links have probability of 0.20.

4) After knowing the failure probability of the underlay link, we could decide the underlay link coverage number for each link according to its failure probability: a) coverage number is 1 if failure probability is between 0 and 0.1; b) coverage number is 2 if failure probability is between 0.1 and 0.2; c) coverage number is 3 if failure probability is larger than 0.2.

5) Then, we use Greedy Algorithm 3 or Optimal Selecting Overlay Links (formula 1 and 2) to find a subset of overlay node links to conduct probing;

6) For each of the failure probability distribution, we will generate 1000 underlay network failure/congestion scenarios using 1000 simulations. Each simulation runs as: a) In each simulation, we will generate random failure/congestion case on each underlay link according to the underlay link's failure/congestion probability α; b) According to the underlay link failure condition, we calculate the corresponding overlay link statuses between the chosen overlay node pairs; c) Based on overlay link statuses, we use Maximum Likelihood Estimation Algorithm 5 or Bayesian Inference Model to infer underlay link statuses. For Bayesian Inference Model, there are two ways to do inference: choose a probability threshold to decide whether failure/congestion happens or do measurement on potential failure/congestion links;

7) Comparing the inferred status with the actual status for each underlay link, we calculate three probabilities: a) Correct Probability: the probability that link inference is correct; b) False Positive Probability: the probability that when the actual link status is on/normal, the inferred status is off/abnormal; c) False Negative probability: the probability that when the actual link status is off/abnormal, the inferred status is on/normal.

**3.2 File description:**

The codes of the MATLAB simulation are all included in the directory "code for MATLAB Simulation ". In that directory, we have one subdirectory named "gt-itm", which contains the files related to GT-ITM topology generator. File "gt-itm-linux.tar" contains source codes of GT-ITM and directory "sample-graphs" includes some sample topology generated by using GT-ITM.

Besides the above described subdirectory, there are lots of ".m" program files in the directory. Those are the core program files and the function of each program file is explained in the following:

* "graphreader.m": read GT-ITM file and randomly add more links to the network topology according to parameter aim\_link\_num. Return the generated graph matrix and link matrix;
* "chooserandomoverlay.m": choose some random overlays according to parameter overlay\_m;
* "exhaustpath.m": according to the chosen overlay, exclude the uncovered underlay links and formulate overlay\_underlay\_matrix and select\_underlay\_matrix;
* "dijkstra.m": according to graph setting, get shortest path route;
* "findlinkindex.m": find the index of the link in the link\_matrix;
* "linkfailureprob.m": given underlay link matrix, generate failure probability of each link following above mentioned three failure probability distributions;
* "generatecoverno.m": generate coverage of each underlay link and the cost of overlay link according to underlay link failure probability and overlay routing condition;
* "greedyalgoselectoverlay.m": Greedy Algorithm 3 to select overlay link;
* "optimalalgoselectoverlay.m": Optimal Selecting Overlay Links (formula 1 and 2) by solving optimization formulations;
* "generatelinkfailurerangeprob.m": generate link failure range matrix;
* "generatelinkfailure.m": according to the failure probability of each link, generate underlay link on/off condition matrix;
* "generateoverlaycondition.m": according to underlay link condition matrix, generate overlay link on/off condition matrix;
* "optimalinferunderlaywithprob.m": use Maximum Likelihood Estimation Algorithm 5 to infer underlay link conditions;
* "bayesianinference.m": use Bayesian Inference Model to infer underlay link conditions;
* "calsame.m": calculate the inference Statistics based on inference results
* "core.m": the main program of the simulation;
* "readonebayesianlogfile.m": get Bayesian inference result of testing K\_num underlay links with largest Bayesian probabilities;
* "printonebayesianlogfile.m": print Bayesian inference result when testing from 1 underlay link to K\_upper underlay links with largest Bayesian probabilities;
* "core1.m": the main program of printing Bayesian inference result when trying to test more underlay links

**3.3 Parameters in the program files**

In the simulation, we will firstly run program "core.m" to get inference result of using Maximum Likelihood Estimation and using Bayesian inference when denoting one probability threshold. Then, we will run program "core1.m" to get inference result of using Bayesian inference when trying to test some number of underlay links. When running the programs, we just need to change the parameters in "core.m" and "core1.m".

The parameters in "core.m" and "core1.m" program files are generalized in the following:

A. The parameters in "core.m":

* "file\_path": the path of GT-ITM network topology configuration file
* "dimension\_m": the number of underlay node
* "aim\_link\_num": the number of underlay link that we try to approach
* "overlay\_m": the number of chosen overlay node
* "threshold\_p": the threshold probability value for Bayesian Inference value

B. The parameters in "core1.m":

* "bayesianlogfile\_name": the path of Bayesian log file generated from core.m
* "total\_underlay\_link\_num": the number of underlay link
* "K\_upper": the upper bound of underlay link numbers that we try to test for Bayesian Inference

**3.4 Explanation of the outputs:**

We will explain the outputs of "core.m" and "core1.m" programs in this section.

A. Explanation of "core.m":

We use one example to explain the output.

One sample of the output of "core.m" is:

" Finally, the underlay link number is 18.

For Loss Probability following Random Distribution:

The chosen overlay link number is 17.

When using Maximum Likelihood Estimation to infer underlay link:

Link same probability is 0.997556, false negative probability is 0.001056, false positive probability is 0.001389;

When using Bayesian Inference Model to infer underlay link:

Link same probability is 0.989833, false negative probability is 0.010167, false positive probability is 0.000000.

For Loss Probability following Normal Distribution:

The chosen overlay link number is 20.

When using Maximum Likelihood Estimation to infer underlay link:

Link same probability is 0.993556, false negative probability is 0.001833, false positive probability is 0.004611;

When using Bayesian Inference Model to infer underlay link:

Link same probability is 0.978944, false negative probability is 0.021056, false positive probability is 0.000000.

For Loss Probability following Step Distribution:

The chosen overlay link number is 17.

When using Maximum Likelihood Estimation to infer underlay link:

Link same probability is 1.000000, false negative probability is 0.000000, false positive probability is 0.000000;

When using Bayesian Inference Model to infer underlay link:

Link same probability is 1.000000, false negative probability is 0.000000, false positive probability is 0.000000. "

In the above output, it lists number of the underlay link finally got. For the loss probability following three different distributions, the output lists the chosen overlay link number and the link inference same probability, link inference false negative probability, link inference false positive probability when using Maximum Likelihood Estimation and Bayesian Inference Model to do inference respectively. Other than these outputs, the program also generate three Bayesian log files named "bayesian\_infer\_1.txt", "bayesian\_infer\_2.txt" and "bayesian\_infer\_3.txt". Each of them corresponds to the log file following one loss probability distribution. The path of the log file is the input parameter "bayesianlogfile\_name" for program file "core1.m" and the underlay link number is the input parameter "total\_underlay\_link\_num" for program file "core1.m".

B. Explanation of "core1.m":

The output plots Bayesian inference result graph when trying to test more underlay links under three different loss probability distributions.

**4. Codes to infer Emulab Experiment Results:**

In the codes to infer Emulab experiment, we parse the overlay measurement results got from Emulab experiments to infer the underlay link conditions. We realize our underlay link inference by using two inference methods: Maximum Likelihood Estimation and Bayesian Inference Model respectively.

**4.1 Code Steps:**

The simulation runs as follows:

1) The data for each experiment consists of three types of files: the probing data file from each node, the link failure configuration file and the routing configuration file. We will parse those data files first, form routing table of overlay nodes, and get probing information among any two different overlay nodes.

2) After forming probing information among two overlay nodes, we will split the time duration of probing into two different types: normal period and abnormal period. Then we will calculate the probing information during normal period and abnormal period. And then we will use the calculated probing information to infer whether congestion in overlay links happens or not.

3) According to the failure probability of the underlay link, we will decide the underlay link coverage number for each link according to its failure probability: a) coverage number is 1 if failure probability is between 0 and 0.1; b) coverage number is 2 if failure probability is between 0.1 and 0.2; c) coverage number is 3 if failure probability is larger than 0.2.

4) Use Greedy Algorithm 3 to find a subset of overlay node links and calculate the related overlay nodes. According to selected overlay nodes, we could calculate the underlay links that are not covered by overlay links.

5) Based on inferred overlay link status, we use Maximum Likelihood Estimation Algorithm 5 or Bayesian Inference Model to infer underlay link status. Here for Bayesian Inference Model, we choose a probability threshold to decide whether failure/congestion happens or not.

6) According to the overlay inference result and real overlay congestion condition, we could calculate Correct Probability, False Positive Probability and False Negative probability. For the inference results got from Maximum Likelihood Estimation Algorithm 5 or Bayesian Inference Model, we could also calculate Unknown Probability, Correct Probability, False Positive Probability and False Negative probability. And these are the inference results for the current overlay nodes setting.

7) If we could further remove overlay node, we will greedily remove the overlay node with the least related overlay links and the program will go to step (4). Otherwise, the program stops.

8) After the program stops, we will generate one result file named analysis\_result, which describes the result of our inference. Then, we will use MATLAB program to print the result graph.

**4.2 Files description:**

The codes of to infer Emulab experiment results are all included in the directory "code for Emulab Experiment". In that directory, we have two subdirectories: one is named "C code" while another one is named "MATLAB code". We will firstly use the codes in "C code" directory to get inference results written in text. Then, we use the codes in "MATLAB code" to plot result graphs by using inference results got from "C code" directory.

In directory "C code/ overlaymeasurementinference", we have one "data" subdirectory, where the files there denotes one set of experiment. For each set of experiment, it includes three types of data: the data in the "data" subdirectory denotes the probing results from each node, the data in file "failuretable" describe the failure probability of each link, the data in file "routing" describe the routing condition. Besides the "data" subdirectory, there are some ".h" and ".c" program files. The functions of those program files are generalized in the following:

* "iterator.c", "iterator.h", "mbcommon.h", "dynarray.c", "dynarray.h": those program files are used for helping generating dynamic array, whose size could be changed dynamically;
* "filereader.c", "filereader.h": the program files for reading configuration files and doing some data reading processing;
* "linkstructure.c", "linkstructure.h": the program files of describing and doing some operatiion about both underlay and overlay link structure;
* "core.c": This is the main file of the program;

In "MATLAB code" directory, we have two program files "core.m", "analysisresultreader.m", which print the results graph by using MATLAB.

The data structures and the functions for above program files are described in the following:

A. For program files "filereader.c", "filereader.h":

* HeartPing: the structure to record each probing item in the data file
* FailureOverlayLink: the structure to record the failure overlay link
* FailureUnderlayLink: the structure to record the failure underlay link
* SingleNodeOverlayMeasurement: the structure to record the measurement result coming from the probing data file of one node
* char \*trim(char \*s): delete the empty space around the char s;
* int get\_node\_index\_from\_char(char \* to\_parsing\_char): parse the node index from the node char;
* void read\_failure\_table\_config\_file(const char \* filename, MBdynarray \*underlaylinkarray): read and parse failure table configuration file;
* void read\_routing\_config\_file(const char \* filename, MBdynarray \*overlaylinkarray, MBdynarray \*underlaylinkarray, double failure\_prob): read and parse routing configuration file;
* int check\_failureoverlylinkarray\_included(MBdynarray \*failureoverlaylinkarray, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array failureoverlaylinkarray. If included, return the index of that item in the array. Otherwise, return -1;
* int check\_failureunderlylinkarray\_included(MBdynarray \*failureunderlaylinkarray, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array failureunderlaylinkarray. If included, return the index of that item in the array. Otherwise, return -1;
* int check\_allnodes\_overlaymeasurement\_array\_included(MBdynarray \*allnodes\_overlaymeasurement\_array, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array allnodes\_overlaymeasurement\_array. If included, return the index of that item in the array. Otherwise, return -1;
* void read\_ping\_result\_file(char\* directory\_name, char \* filename, int \*min\_start, int \*max\_start, int \*min\_end, int \*max\_end, MBdynarray \*singlenodepingarrary, MBdynarray \*failureoverlaylinkarray): read and parse probing result file;
* void read\_ping\_result\_directory(char \* directory\_name, int \*min\_start, int \*max\_start, int \*min\_end, int \*max\_end, MBdynarray \*failureoverlaylinkarray, MBdynarray \*allnodes\_pingarrary): read and parse probing result files in the probing result directory;
* void parsing\_ping\_result(int \*min\_start, int \*max\_start, int \*min\_end, int \*max\_end, MBdynarray \*failurelinkarray, MBdynarray \*allnodes\_pingarrary, MBdynarray \*allnodes\_overlaymeasurement\_array): parsing the ping result got from overlay measurement array;
* void print\_delay\_array(MBdynarray \*arr): print the integer value of delay array;
* void print\_singlenode\_overlayMeasurement(SingleNodeOverlayMeasurement \*singlenode\_overlayMeasurement): print overlay measurement array of single node;
* void print\_allnodes\_overlayMeasurement(MBdynarray \*allnodes\_overlaymeasurement\_array): print overlay measurement array of all nodes;
* void print\_selected\_overlayMeasurement(MBdynarray \*allnodes\_overlaymeasurement\_array, MBdynarray \*overlaylinkarray): print selected overlay measurement array of all nodes;
* MBdynarray \* get\_failure\_underlay\_link\_array(MBdynarray \*failure\_overlay\_link\_array, MBdynarray \*overlaylinkarray): get the print the condition of failure underlay link array;
* void print\_failure\_overlay\_link(FailureOverlayLink \*failure\_overlay\_link\_item): print the condition of failure overlay link item;
* void print\_failure\_overlay\_link\_array(MBdynarray \*failure\_overlay\_link\_array): print the condition of failure overlay link array;
* void print\_failure\_underlay\_link(FailureUnderlayLink \*failure\_underlay\_link\_item): print the condition of failure underlay link item;
* void print\_failure\_underlay\_link\_array(MBdynarray \*failure\_underlay\_link\_array): print the condition of failure underlay link array;
* void print\_allnodes\_pingarray(MBdynarray \* allnodes\_pingarrary): print the condition of the all nodes ping array;
* void parsing\_overlaymeasurement\_result(MBdynarray \*allnodes\_overlaymeasurement\_array, MBdynarray \*overlaylinkarray): parsing the result of overlay link measurement and decide whether overlay link item fails or not;
* void bubbleSort(MBdynarray \* arr): rank the array arr from small to large according to the item value;

B. For program files "linkstructure.c", "linkstructure.h":

* Underlaylink: the data structure of recording the condition of one underlay link;
* Overlaylink: the data structure of recording the condition of one overlay link;
* OverlayNode: the data structure of recording the behavior of one overlay node;
* Overlaylink \* default\_overlaylink(): construct default values for Overlaylink structure;
* Underlaylink \* default\_underlaylink():construct default values for Underlaylink structure;
* Overlaylink \* init\_overlaylink(int src\_index, int dst\_index, int total\_index): init some values for Overlaylink structure;
* void print\_underlay\_link(Underlaylink \*underlay\_link\_item): print the condition of one underlay link item;
* void print\_underlay\_link\_array(MBdynarray \*underlay\_link\_array): print the condition of the underlay link array;
* int check\_array\_included(MBdynarray \*sorted\_underlay\_index, int inserted\_i): find if inserted\_i is included in sorted\_underlay\_index. If it is included in the array, return the index of that item in the array. Otherwise, return -1;
* void insert\_sorted\_array(MBdynarray \*potential\_underlay\_index, int inserted\_i): insert item inserted\_i into array potential\_underlay\_index so that the array is still sorted;
* int check\_failure\_underlayarray\_included(MBdynarray \*failureunderlaylinkarray, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array failureunderlaylinkarray. If included, return the index of that item in the array. Otherwise, return -1;
* void cal\_underlay\_link\_array\_result(FILE \* result\_file\_fp, MBdynarray \*failureunderlaylinkarray, MBdynarray \*underlay\_link\_array, MBdynarray \*overlaylinkarray, MBdynarray \*uncovered\_underlay\_array): calculate the statistics for the result of link inference;
* void print\_overlay\_link(Overlaylink \*overlay\_link\_item): print the condition of one overlay link item;
* void print\_overlay\_link\_array(MBdynarray \*overlay\_link\_array): print the condition of one overlay link array;
* int check\_overlayarray\_included(MBdynarray \*overlaylinkarray, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array overlaylinkarray. If included, return the index of that item in the array. Otherwise, return -1;
* void change\_underlayLink\_failure\_prob(Underlaylink \* underlaylinkitem, double failure\_prob): change the failure probability of the underlay link item underlaylinkitem;
* Underlaylink \* init\_underlaylink(int src\_index, int dst\_index, int total\_index): init some values for Underlaylink structure;
* int check\_underlayarray\_included(MBdynarray \*underlaylinkarray, int src\_index, int dst\_index): check if (src\_index, dst\_index) is included in array underlaylinkarray. If included, return the index of that item in the array. Otherwise, return -1;
* void generate\_underlay\_coverage\_num(MBdynarray \*underlaylinkarray): generate the coverage number for each underlay link items in array underlaylinkarray;
* void cal\_overlay\_cost(MBdynarray \*overlaylinkarray): calculate the cost for each overlay link items in array overlaylinkarray;
* int check\_covered\_underlay\_array\_included(MBdynarray \*covered\_underlay, int underlaylink\_index): check if underlaylink\_index is included in array covered\_underlay. If included, return the index of that item in the array. Otherwise, return -1;
* MBdynarray \*get\_overlay\_node\_array(MBdynarray \*selected\_overlayLink\_array): according to the array of overlay link, get the corresponding overlay nodes array;
* void get\_uncovered\_underlay\_array(MBdynarray \*selected\_overlayLink\_array, MBdynarray \*underlaylinkarray, MBdynarray \*uncovered\_underlay\_array): get the array of underlay links that is not covered by the current overlay links;
* MBdynarray \*greedy\_select\_overlay(MBdynarray \*overlaylinkarray, MBdynarray \*underlaylinkarray, MBdynarray \*noncoverednodearray): Greedy Algorithm 3 to find the overlay links;
* void print\_overlay\_node\_item(OverlayNode \*overlay\_node\_item): print the condition of one overlay node item;
* void print\_overlay\_node\_array(MBdynarray \*overlay\_node\_array): print the condition of the overlay node array;
* void print\_integer\_nodearray(MBdynarray \*integernodearray): print the integer item in the array;
* MBdynarray \*greedy\_select\_overlay\_exclude\_one\_item(MBdynarray \*selected\_overlayLink\_array, MBdynarray \*overlaylinkarray, MBdynarray \*underlaylinkarray, MBdynarray \*noncoverednodearray): greedily remove the overlay node with the least related overlay links and then use Greedy Algorithm 3 to find the overlay links;
* int check\_removedoverlayindexarray\_included(MBdynarray \*removed\_overlay\_index\_array, int overlay\_index): check if overlay\_index is included in array removed\_overlay\_index\_array. If included, return the index of that item in the array. Otherwise, return -1;
* int check\_noncoverednodearray\_included(MBdynarray \*noncoverednodearray, int node\_index): check if node\_index is included in array noncoverednodearray. If included, return the index of that item in the array. Otherwise, return -1;
* int check\_overlaynodearray\_included(MBdynarray \*overlay\_node\_array, int node\_index): check if node\_index is included in array overlay\_node\_array. If included, return the index of that item in the array. Otherwise, return -1;
* MBdynarray\* cal\_overlayNode\_info(MBdynarray \*selected\_overlayLink\_array): according to overlay link condition, calculate conditions of overlay node;
* int find\_lowest\_overlay\_node\_array\_index(MBdynarray \*overlay\_node\_array): find the lowest index in overlay node array;
* MBdynarray \* get\_complement\_set\_without\_two(MBdynarray \*sorted\_underlay\_index, MBdynarray \*uncovered\_underlay\_array, int upper\_len): get the complement array set of sorted\_underlay\_index and uncovered\_underlay\_array;
* void get\_complement\_set(MBdynarray \*sorted\_underlay\_index, int upper\_len, MBdynarray \*complement\_set): get the complement array set of sorted\_underlay\_index;
* void greedy\_infer\_underlay(MBdynarray \*selectedoverlaylinkarray, MBdynarray \*underlaylinkarray, MBdynarray \*uncovered\_underlay\_array): Maximum Likelihood Estimation Algorithm 5 to infer the underlay conditions;
* void get\_all\_comb\_binary\_matrix(int total\_amount, int amount\_0, MBdynarray \* all\_comb\_matrix): get the possible binary combination array if there exists amount\_0 zero numbers out of total\_amount numbers;
* void bayesian\_infer\_underlay\_cal(MBdynarray \*selectedoverlaylinkarray, MBdynarray \*underlaylinkarray, MBdynarray \*uncovered\_underlay\_array): use Bayesian Inference Model to infer the underlay link conditions;
* void bayesian\_underlay\_link\_array\_infer\_use\_bound(MBdynarray \*underlay\_link\_array, double bayesian\_prob\_bound): according to the probability bound value bayesian\_prob\_bound to decide whether failure happens or not.

**4.3 Parameters in the program files:**

We only need to change parameters in "core.c" and "core.m" when running the program. The parameters are generalized in the following:

A. parameters in "core.c":

* "result\_file\_name": the path of the result for final analysis;
* "routing\_config\_file\_name": the path of routing configuration file;
* "ping\_directory\_name": the path of the probing result directory;
* "failure\_config\_file\_name": the path of failure configuration file table;
* "bayesian\_prob\_bound": the Bayesian probability bound value to infer whether failure happens or not.

B. parameters in "core.m":

* "file\_path": the path of analysis result;

**4.4 Explanation of outputs:**

We will explain the outputs when using code to infer result got from Emulab experiment.

A. Explanation of "core.c":

All the parsing result is formed in the file named "analysis\_result". Every five lines form one basic unit of the output. One unit of the output is shown like below:

" 29 71

0.746479 0.112676 0.140845

0.888889 0.000000 0.111111 0.000000

0.888889 0.000000 0.111111 0.000000

0.888889 0.000000 0.111111

0.888889 0.000000 0.111111"

For the first line, "29" means number of overlay node and "71" means number of overlay links. For the second line, "0.746479 0.112676 0.140845" are describing same probability, false negative probability and false positive probability for the overlay link inference. The third and forth lines

"0.888889 0.000000 0.111111 0.000000

0.888889 0.000000 0.111111 0.000000"

are describing same probability, false negative probability, false positive probability, unknown probability for underlay link inference by using Maximum Likelihood Estimation Algorithm 5 and Bayesian Inference Model respectively. The fifth and sixth lines

"0.888889 0.000000 0.111111

0.888889 0.000000 0.111111"

are describing same probability, false negative probability, false positive probability for underlay link inference by using Maximum Likelihood Estimation Algorithm 5 and Bayesian Inference Model respectively when without considering unknown probability.

B. Explanation of "core.m":

The program will generate six types of result graph:

* Chosen overlay link number graph
* Overlay link inference result graph
* Underlay link inference result using Maximum Likelihood Estimation graph
* Underlay link inference result using Bayesian Inference Model graph
* Underlay link inference result using Maximum Likelihood Estimation graph (not considering unknown probability)
* Underlay link inference result using Bayesian Inference Model graph (not considering unknown probability)

**5. Conclusion:**

In conclude, we introduced the codes of our simulation to do multi-resolution overlay measurement.