# ComplexCi

* **Introduction**

This project (ComplexCi) mainly focuses on the c++ implementation of Collective Influence (CI) algorithm, which is designed to find the most important node (or influencers) in the Complex Network via optimal percolation developed by

Morone F, Makse H A. Influence maximization in complex networks through optimal percolation[J]. Nature, 2015, 524(7563): 65-U122.

Morone F, Min B, Bo L, et al. Collective Influence Algorithm to find influencers via optimal percolation in massively large social media[J]. Scientific reports, 2016, 6.

Overall, the target of CI algorithm is to give a ranking list of nodes according to their importance and the top-ranked nodes will have more importance. We can remove the nodes from the top-ranked ones in the ranking list generated by CI algorithm and calculate the size of giant component after each removal, which will break down the network into many disconnected pieces. The ratio of giant component will reach zero with the one-by-one removal operation finally. Therefore, the better algorithm, the sooner the network will collapse to the zero giant component with smaller count of provided nodes.

Users can get the minimal set of influencers of the Complex Network by the C++ program in this repository . Considered that CI algorithms are able to be used in so many scientific fields , I implement this algorithm in modern C++ style code. Compared with the original c code <http://www-levich.engr.ccny.cuny.edu/~hernanlab/uploads/CI_HEAP.c> mentioned on the http://www-levich.engr.ccny.cuny.edu/webpage/hmakse/software-and-data/ with above paper , the ComplexCi has the following features :

1. This project is using the c++ code style of modern oriented object programming with Standard Template Library (STL), which is easier for users to modify than the original c code. After reviewing the original c language code CI\_HEAP.c, I believe it is difficult for users to read , maintain and implement it if they have their own idea, especially CI\_HEAP.c contains lots of simple variable abbreviation, multilevel pointer, unsafe memory management and unfriendly c style code.
2. ComplexCi accepts more input parameters and they can be used in more flexible behavior of operating nodes in the provided network.
   1. Users can determine the batch size of deleting nodes in ComplexCi per updating CI values when the Complex Network collapses
   2. Users can determine the certain giant component ratio when starting re-inserting algorithm which complex network collapses to
   3. Users can determine whether they need re-insert operation in the CI algorithm to optimize the result
3. Verified that the result of traditional c implementation and new c++ can both achieve the nearly same result in the metric of Robustness, even the new c++ implementation is more efficient and spends much less time on some datasets than the traditional c program. See in the Benchmark Section
4. The traditional CI\_HEAP.c is merged into this project as well. Users can switch the option to use the new c++ designer or the traditional c style code depending on their taste.

* **Usage**

This section describes the Usage of the ComplexCi and its corresponding scripts

* + **Get** **Repository**
    - **Download Release**
    - **Fetch from Github Source**

Users can directly clone the repository by the git command or just download the zip archiver on the webpage

git clone https://github.com/zhfkt/ComplexCi.git

* + - * **Compile**

There are lots of C++11 features and syntax in the code so that the C++ Compiler needs to support C++11 Standard . In fact, there is only one cpp file ComplexCi/ComplexCi.cpp need to be compiled and it doesn’t rely on the other extra library

* + - * + **Linux**

Users can enter into the “bin/” under root project folder and execute

g++ -pthread ../ComplexCi/ComplexCi.cpp -o ComplexCi -O3 -std=c++11 ;

or just execute

./make.sh

to generate binary bin/ComplexCi . Pls notice that the version of g++ needs to support c++11. For my own dev, the compilation is passed under g++ 5.4.0 on Ubuntu 16.04. Otherwise , you will get failure of several incorrect syntax errors.

* + - * + **Windows**

There has already been a windows binary file x64/Release/ComplexCi.exe compiled on the Windows7 x64 in the repository. It can just be used if users do not need to recompile or change the code. If users want to compile themselves and have the Visual Studio 2013 or higher version, they can directly open the ComplexCi.sln and compile the code in the IDE. The binary file will be generated under x64/Release/ComplexCi.exe . I believe users can also pick up any other IDE or Compiler supporting C++11 to compile the file ComplexCi/ComplexCi.cpp . Pls note that replace the x64/Release/ComplexCi.exe with your own generate file after compilation.

* + **Run**
    - **Network Input File Format**

The data of network can be written in one txt/csv files and the network is considered as undirected network. Each row contains 2 node IDs divided by one comma, which means there is a connection between these 2 nodes. The node IDs should be integer and started from 0. For example:

0,1

1,2

1,0

2,1

....

The complete Example is at data/test/karate.csv

* + - **Easy Start to use Script**

Using Script in the project is a quick start to utilize Collective Influence (CI) algorithm. There are 3 scripts users can execute in the project.

1. traditionalCollectiveInfluence: Users can use this script to utilize traditional c style code of <http://www-levich.engr.ccny.cuny.edu/~hernanlab/uploads/CI_HEAP.c> . I merge it into this repository and set the compatible interface layer to call the c code in c++ .

Script “traditionalCollectiveInfluence” accepts 3 parameters:

./traditionalCollectiveInfluence.sh <networkPath> <ballRadius> <isPrintMinPointCausingMinComponent>

<networkPath> is the file path. The file format is described in the section Network Input File Format

<ballRadius> is the Radius parameter defined in the Collective Influence Algorithm

When the ballRadius is zero, pls notice that Collective Influence Algorithm will degenerate into HDA (high degree adaptive) algorithm. a.k.a. CI value of each node will be equal with degree of the node.

<isPrintMinPointCausingMinComponent> whether the output file contains limited point leading to 0.01 of giant component ratio or all points. If it is set to 0, the program will output all nodes. Otherwise the program will output partial points, which will make the giant component ratio to 0.01 in the deleting nodes process

e.g.

./traditionalCollectiveInfluence.sh /home/network/model1.csv 3 1  
  
 It means users are using the traditionalCollectiveInfluence strategy for the input file model1.csv with parameter ballRadius 3. The output file only contain the partial points, which will make the giant component ratio to 0.01 in the deleting nodes process

1. cppCollectiveInfluence: Users can use this script to utilize new c++ implementation of Collective Influence (CI) algorithm.   
     
   ./cppCollectiveInfluence.sh <networkPath> <ballRadius> <updateBatch> <isPrintMinPointCausingMinComponent>

<networkPath> is the file path. The file format is described in the section Network Input File Format

<ballRadius> is the Radius parameter defined in the Collective Influence Algorithm

When the ballRadius is zero, pls notice that Collective Influence Algorithm will degenerate into HDA (high degree adaptive) algorithm. a.k.a. CI value of each node will be equal with degree of the node.

<updateBatch> batch size of deleting nodes in Network per updating CI values when the Complex Network collapses

<isPrintMinPointCausingMinComponent> whether the output file contains limited point leading to 0 of giant component ratio or all points. If it is set to 0, the program will output all nodes. Otherwise the program will output partial points, which will make the giant component ratio to 0 in the deleting nodes process. i.e. There is no edge but still left point in the network. Pls notice the different behavior compared with traditionalCollectiveInfluence

e.g.

./cppCollectiveInfluence.sh /home/network/model1.csv 2 500 0

It means users are using the cppCollectiveInfluence strategy for the input file model1.csv with parameter ballRadius 2. The output file only contain all nodes and 500 nodes will be deleted in a batch per updating CI values when the Complex Network collapses.

1. newReinsertCollectiveInfluence: This script involves the **new re-insert algorithm** of collective influence. After verified on the 8 datasets, this new algorithm can achieve better performance in the metric of Robustness than **original re-insert algorithm** in collective influence. See in the benchmark sections.  
     
   ./newReinsertCollectiveInfluence.sh <networkPath> <ballRadius> <updateBatch> <isPrintMinPointCausingMinComponent>

<networkPath> is the file path. The file format is described in the section Network Input File Format

<ballRadius> is the Radius parameter defined in the Collective Influence Algorithm

When the ballRadius is zero, pls notice that Collective Influence Algorithm will degenerate into HDA (high degree adaptive) algorithm. a.k.a. CI value of each node will be equal with degree of the node.

<updateBatch> batch size of deleting nodes in Network per updating CI values when the Complex Network collapses

<isPrintMinPointCausingMinComponent> whether the output file contains limited point leading to 0 of giant component ratio or all points. If it is set to 0, the program will output all nodes. Otherwise the program will output partial points, which will make the giant component ratio to 0 in the deleting nodes process. i.e. There is no edge but still left point in the network. Pls notice the different behavior compared with traditionalCollectiveInfluence

e.g.

./newReinsertCollectiveInfluence.sh /home/network/model1.csv 2 500 0

It means users are using the newReinsertCollectiveInfluence strategy for the input file model1.csv with parameter ballRadius 2. The output file only contain all nodes and 500 nodes will be deleted in a batch per updating CI values when the Complex Network collapses.

The project contains both bash scripts for linux and cmd scripts for windows. Bash for linux can be found in the dailyUse\linux and cmd for windows are in the dailyUse\windows

* + - **Directly use with Binary ComplexCi**If users want to control the more flexible behaviour in the Collective Influence algorithm, binary program ComplexCi is able to achieve the goal.  
        
      ./ComplexCi [path] [ballRadius] [updateBatch] [outputNumBatch] [method] [biggestComponentEndThreshold] [isPrintMinPointCausingMinComponent]

<path> is the file path. The file format is described in the section Network Input File Format

<ballRadius> is the Radius parameter defined in the Collective Influence Algorithm

When the ballRadius is zero, pls notice that Collective Influence Algorithm will degenerate into HDA (high degree adaptive) algorithm. a.k.a. CI value of each node will be equal with degree of the node.

<updateBatch> batch size of deleted points per updating Collective Influence value (traditional ci will be fixed to 1). Users can determine the batch size of deleting nodes in ComplexCi per updating CI values when the Complex Network collapses

<outputNumBatch> output number of point each line in the final result file

<method> method of several algorithms in collective influence. Here users can input the integer represented as method

1. 0 means the new c++ implementation of Collective Influence (CI) algorithm **without** re-insert process at the end
2. 1 means the traditional c style implementation (<http://www-levich.engr.ccny.cuny.edu/~hernanlab/uploads/CI_HEAP.c>) of Collective Influence (CI) algorithm **with original** re-insert process at the end. This method is also called by the script **traditionalCollectiveInfluence** mentioned above
3. 2 means the traditional c style implementation (<http://www-levich.engr.ccny.cuny.edu/~hernanlab/uploads/CI_HEAP.c>) of Collective Influence (CI) algorithm **without** re-insert process at the end
4. 3 is not used here.
5. (Experimental) 4 means c++ **concurrent** implementation of Collective Influence (CI) algorithm **without** re-insert process at the end (method 1). However , the algorithm is not mature and even sometimes it will be slower than single thread for the small dataset. Because forking thread is an expensive operation on the operation system level and there is no internal thread pool in c++.
6. 5 means the new c++ implementation of Collective Influence (CI) algorithm **with original** re-insert process at the end using **more strict** internal parameters. In detail , there are 2 internal parameters changed here.
   1. computeComponentInterval: It means the Interval to calculate Giant Component when Complex Network Collapses. The original is 1 Percent of the size of complete network. For the more strict parameter , the value is re-scaled to 200 if the original is larger than 200.
   2. reinsertEachStep: It means the batch size of nodes reinserted into the network per updating the graph. The original is 1 Per mille of the size of complete network. For the more strict parameter , the value is re-scaled to 20 if the original is larger than 20.
7. (Experimental) 6 means c++ **concurrent** implementation of Collective Influence (CI) algorithm **with original** re-insert process at the end in **more strict** internal parameters (method 5). However , the algorithm is not mature and even sometimes it will be slower than single thread for the small dataset. Because forking thread is an expensive operation on the operation system level and there is no internal thread pool in c++.
8. 7 means the new c++ implementation of Collective Influence (CI) algorithm **with new** re-insert process at the end using **more strict** internal parameters. In detail , there are 2 internal parameters changed here.
   1. computeComponentInterval: It means the Interval to calculate Giant Component when Complex Network Collapses. The original is 1 Percent of the size of complete network. For the more strict parameter , the value is re-scaled to 200 if the original is larger than 200.
   2. reinsertEachStep: It means the batch size of nodes reinserted into the network per updating the graph. The original is 1 Per mille of the size of complete network. For the more strict parameter , the value is re-scaled to 20 if the original is larger than 20.
9. (Experimental) 8 means c++ **concurrent** implementation of Collective Influence (CI) algorithm **with new** re-insert process at the end in **more strict** internal parameters (method 7). However , the algorithm is not mature and even sometimes it will be slower than single thread for the small dataset. Because forking thread is an expensive operation on the operation system level and there is no internal thread pool in c++.
10. 9 means the new c++ implementation of Collective Influence (CI) algorithm **with original** re-insert process at the end using **the same** internal parameters. This method is also called by the script **cppCollectiveInfluence** mentioned above. In detail , there are 2 same internal parameters here.
    1. computeComponentInterval: It means the Interval to calculate Giant Component when Complex Network Collapses. The original is 1 Percent of the size of complete network.
    2. reinsertEachStep: It means the batch size of nodes reinserted into the network per updating the graph. The original is 1 Per mille of the size of complete network.
11. 10 means the new c++ implementation of Collective Influence (CI) algorithm **with new** re-insert process at the end using **the same** internal parameters. This method is also called by the script **newReinsertCollectiveInfluence** mentioned above. In detail , there are 2 same internal parameters here.
    1. computeComponentInterval: It means the Interval to calculate Giant Component when Complex Network Collapses. The original is 1 Percent of the size of complete network.
    2. reinsertEachStep: It means the batch size of nodes reinserted into the network per updating the graph. The original is 1 Per mille of the size of complete network.
12. (Experimental) 11 is based on method 6 . Another enhanced re-insert algorithm is applied on it ,which will choose rank **behind** in the sequence of node importance with high priority. Method 6 will just ignore the rank information. This method is also the **concurrent** implementation of the method 15.
13. (Experimental) 12 is based on method 8 . Another enhanced re-insert algorithm is applied on it ,which will choose rank **behind** in the sequence of node importance with high priority. Method 8 will just ignore the rank information. This method is also the **concurrent** implementation of the method 16.
14. (Experimental) 13 is based on method 8. Reinsert algorithm will use the **multiple** kernel when combining giant component information . This method is also the **concurrent** implementation of the method 17.
15. (Experimental) 14 is based on method 13. Reinsert algorithm will choose rank **behind** in the sequence of node importance with high priority combined with **multiple** kernel. This method is also the **concurrent** implementation of the method 18.
16. (Experimental) 15 is based on method 6 . Another enhanced re-insert algorithm is applied on it ,which will choose rank **behind** in the sequence of node importance with high priority. Method 6 will just ignore the rank information. This method is also the **single** thread implementation of the method 11.
17. (Experimental) 16 is based on method 8 . Another enhanced re-insert algorithm is applied on it ,which will choose rank **behind** in the sequence of node importance with high priority. Method 8 will just ignore the rank information. This method is also the **single** thread implementation of the method 12.
18. (Experimental) 17 is based on method 8. Reinsert algorithm will use the **multiple** kernel when combining giant component information . This method is also the **single** implementation of the method 13.
19. (Experimental) 18 is based on method 17. Reinsert algorithm will choose rank **behind** in the sequence of node importance with high priority combined with **multiple** kernel. This method is also the **single** thread implementation of the method 14.

I believe users can just use the method without Experimental tag for their daily use because these methods with Experimental tag do not take effect obviously for most Complex Network datasets. There is no need for users to get the knowledge of these method in most cases. Pls ignore them. See in the DataCastle Comptition Section.

<biggestComponentEndThreshold> complex network collapses to the certain giant component ratio where the reinsert algorithm starts for cpp ci . Traditional ci will be fixed to 0.01 . If the method 0, 2, 4 are used, <biggestComponentEndThreshold> will not be used without Reinsertion

<isPrintMinPointCausingMinComponent> whether output limited point leading to 0(new cpp ci)/0.01(traditional ci) of giant component ratio or all points. If it is set to 0, the program will output all nodes. Otherwise the program will output partial nodes. This input parameter contains the same meaning with the <isPrintMinPointCausingMinComponent> in the script traditionalCollectiveInfluence, cppCollectiveInfluence and newReinsertCollectiveInfluence.

If users compiles correctly , the binary program will be in the bin/ComplexCi on the linux platform and x64/Release/ComplexCi.exe on the windows platform.

* + - **Output file**

When the algorithm finishes, the output file will be generated in the same folder of the original file with suffix “\_out”. Each row in the output file contains nodes, whose count is a group of <outputNumBatch> defined in running ComplexCi binary. If users runs the script, the value <outputNumBatch> will be fixed to 500, for instance：

model1,4,5,2,33,…,2214

model1,2432,4554,3222,1123,…,2233

…

From the above instance, the nodes of networks model1 will be removed following such order: 4, 5, 2, 33, …, 2214, 2432, 4554, 3222, 1123, …, 2233… . If nodes count % outputNumBatch !=0, some comma will be at the last line of the output.

* + **Benchmark**

In order to demonstrate the performance of 3 main algorithms mentioned in the scripts traditionalCollectiveInfluence, cppCollectiveInfluence and newReinsertCollectiveInfluence, I provide 8 test datasets downloaded from DataCastle Competition at GITHUB LINK . Here the metric of Robustness is another measure to quantify the performance of ranking methods introduced by the paper

Schneider C M, Moreira A A, Andrade J S, et al. Mitigation of malicious attacks on networks[J]. Proceedings of the National Academy of Sciences, 2011, 108(10): 3838-3841.

The smaller value is, the better the algorithm is.

BenchMark TABLE

From the benchmark ,we can see that the result of traditional c implementation traditionalCollectiveInfluence and new c++ cppCollectiveInfluence can both achieve nearly the same result in the metric of Robustness, even the new c++ implementation is more efficient and spends much less time on some datasets than the traditional c program. Data structure of disjoint-set is used in the reinsertion in the new c++ implementation ComplexCi and it can boost a lot. The traditionalCollectiveInfluence didn’t use this data structure and I think that’s the reason why the traditional c program was slow.

We can also see that newReinsertCollectiveInfluence can achieve better Robustness Value result. It can be proved that the newReinsertCollectiveInfluence performs well even the ballRadius is set to 0 in the simple HDA (high degree adaptive) algorithm without using Collective Influence algorithm.

* + **DataCastle** **Competition**
    - **Introduction**<http://www.pkbigdata.com/common/cmpt/%E5%A4%A7%E5%B8%88%E8%B5%9B_%E7%AB%9E%E8%B5%9B%E4%BF%A1%E6%81%AF.html?lang=en_US>  
      Competition background

Disparate networks, including social networks, communication networks and biological networks, are playing an increasingly important role on natural and socio-economic systems. A core problem, therein, is to measure the significance of individual nodes. For instance, a super spreader in Hong Kong triggered transmission of SARS to a significantly greater number of other people than 100 normal infected persons;a rumor re-tweeted by a celebrity may spread much broader than that by an obscure person.

Therefore it is necessary to develop a method to identify thevirulence genes in large-scale gene regulatory networks, to find the super-spreaders in large-scale social networks, and to detect the key enterprises with serious systematic financial risk in large-scale financial networks.

Those tasks could be formalized as a generic challenge that is identifying vital nodes in networks that are important for sustaining connectivity. This challenge, aka optimal percolation, is a well-documented issue in network science. With great anticipation of making big progress on this problem, we successfully invited some experts and hope the great participants will create novel and effective solutions.

* + - **How to get Quick/Best Result**

Here I write 2 scripts both on the windows and linux platform to help users to get the quick and best results.   
 The quickresult script is a quick way to generate the output. Compared with the best result, it doesn’t spend too much time and the result is still competitive and suprisingly in the scale of million. The quickresult script can achieve NUM in the metric of Robustness.   
 The bestresult script is to get the score of NUM in the best performance in terms of Robustness. However, it will take much longer time to finish in serveral hours.

If users want to experience the quick/best result of the raised algorithm for DataCastle Competition, they can follow the steps:

1. Download zip files networks.zip containing 8 datasets from GITHUB LINK and extract it under data/networks/ . That’s say, the 8 dataset files will be  
     
    data/networks/model1.csv

data/networks/model2.csv

data/networks/model3.csv

data/networks/model4.csv

data/networks/real1.csv

data/networks/real2.csv

data/networks/real3.csv

data/networks/real4.csv

1. Compile the code according to the section “Compile” on windows/linux or just use the current binary file under windows in the Release zip

3. For the linux on the 4-core machine, users can execute the **quickResult.sh** under bin folder to get a quick result and wait until finish in about **5 minutes** in serveral million-scale networks. Users are also able to execute the **bestResult.sh** under bin folder to get the best result but it will take nearly 8 hours to complete the whole algorithm for all datasets.

For the windows, users can execute the **quickResult.cmd** under x64/release/ folder and the cmd will pop up 8 separate consoles. Until these consoles all complete the algorithm, users are able to use the **mergeResult.cmd** to merge the 8 files into one result file. For the **bestResult.cmd,** users can execute it to get a best result, but which will take longer time than linux version because it is not implemented in parallel for 8 datasets.

Output results will be in a folder under data/networks/results/ for both windows and linux

* + - **Monitor** **Quick/Best Result on the linux environment**

Due to the reason that Quick/Best Result script run in background on the linux environment, users can monitor the background job by the command in the bin folder:

tail -f <serID>.log

<serID> will be shown on the script running screen

* + - **Detailed algorithm**

I will describe detailed algorithm in the experiment paper soon.

* + **Folder/Files**

ComplexCi:

ComplexCi/ComplexCi.cpp contains main algorithm for Collective Influence Algorithm

ComplexCi/CI\_HEAP.c contains main algorithm for Collective Influence Algorithm fetched from <http://www-levich.engr.ccny.cuny.edu/~hernanlab/uploads/CI_HEAP.c> . I slighlty modify the code in order to make it compatiable with C++ compiler.

ComplexCiPython:

Python code of other algorithms to verify the dataset from DataCastle as the compartive algorithm.

Master\_algorithm:

Evaluation program in verifying 8 datasets in the DataCastle Competition which is grabbed from <http://share.pkbigdata.com/ID.4407/Master_algorithm> .

Bin:

Contains lots of various linux scripts to be used in the different behaivour of controling testing, compiling and running.  
 quickResult.sh is the bash script to generate the quick result of DataCastle Competition for 8 datasets

bestResult.sh is the bash script to generate the best result of DataCastle Competition for 8 datasets

dailyUse:

Contains 3 scripts of main algorithms for both linux bash and windows cmd

data/test:

Some test input datasets.

x64/Release:  
 x64/Release/ComplexCi.exe is the Windows binary file compiled on Windows 7 x64

x64/Release/quickResult.cmd is the cmd script to generate the quick result of DataCastle Competition for 8 datasets

x64/Release/bestResult.cmd is the cmd script to generate the best result of DataCastle Competition for 8 datasets

ComplexCi.sln

Visual Studio 2013 Project file of ComplexCi.