

Part II K-Cliques Counting Algorithms With DOULION: Implementation And Experiments

Abstract:

Clique problem becomes more and more important in nowadays. However, as the target graphs becomes larger, solutions become unavailable because clique problems are mostly NP-complete. This part aims to combine DOULION algorithm with k cliques counting algorithms to investigate the properties of this innovative method. Two k-cliques counting algorithms are successfully implemented and the correctness are verified. After comparing the performance of two implementations, we choose Bron's Algorithm for further research. Several different types of graphs are used for combination of Bron's Algorithm and DOULION algorithm. Results show that DOULION algorithm could help speed up the calculation when not decrease the quality of counting result, yet the stability need to be improved by repeated experiments.

Introduction

In graph theory, the clique problem is any problems that relate to find complete subgraphs, or cliques in a graph [1]. There are many applications involving in clique problems, including social networks, bioinformatics, and computational chemistry. Many problems involve in such field, including finding a maximum clique; finding maximum weight clique in weighted graph, listing all maximal cliques, and decision problem of testing whether a graph has a clique larger than given size. Unfortunately all above problems are NP-complete, which is one of Karp's 21 NP-complete problems [2]. The focus of the two algorithms we implement here on maximum cliques finding and maximal cliques listing requires exponential time, however, could avoid unnecessary calculations.

Previously, we investigated triangle-counting problem using DOULION preprocessing algorithm with normal counting methods [7]. Essentially it is also a k-clique problem, whose k is equal to 3. Tsourakakis, Charalampos E., et al. already figured out the efficiency and correctness when k is equal to 3 using DOULION method [7]. However, it is unclear and becoming complicate when k increases in time and memory costing, as well as correctness and efficiency, although k-clique problem shares many similarities within various k values. In this part, we will implement two algorithms involving listing maximal k-cliques problem, and investigate the effects of DOULION algorithm on them for different types of graphs in experiments.

Before that, we need to figure out terminologies related to these problems. A maximal clique, also called inclusion-maximal, is a clique that is included in a larger clique, which cannot be enlarged by the addition of any vertex in the graph; while a maximum clique is the maximum maximal clique that is not include by a larger clique. K-clique counting problem is related to above two problem: we need to decide the maximum k in the graph as well as how many cliques when k value various in the same graph. As *Moom J.W. et al.* claimed in 1965 [3], any n -vertex graph has at most $3^{n/3}$ maximal cliques. The Bron-Kerbosch algorithm [4] (short for Bron's Algorithm in this report) is basically a recursive backtracking procedure by increasing a candidate cliques when they consider one vertex one time, and either add it to candidate or excluded vertex sets, which has a worst-case running time $O(3^{n/3})$. However, Bron's Algorithm is reported faster in practice than other alternative algorithms, reported by Cazals & Karande in 2008 [5]. As a comparison, another algorithm, invented by Yun Zhang [6], will be implemented also (short for Yun's Algorithm in this report).

Bron's Algorithm is one of the classic and most popular algorithms to solve maximal k -cliques problem [4]. Essentially, it is a recursive backtracking strategy to solve maximal k -cliques problem viewed as a depth-first traversal search tree on three dynamically changing sets: 1) COMPSUB, a global set containing the clique-in-progress; 2) CANDIDATES, a local set containing all vertices that will be added to the current COMPSUB; and 3) NOT, a local set containing all vertices that have been previously added to COMPSUB. Initially, COMPSUB and NOT are empty. In each step, a vertex v is removed from CANDIDATES to COMPSUB to generate new set NEW_CANDIDATES, which is the intersection of CANDIDATES and neighborhood of v . When both NEW_CANDIDATES and NOT are empty, a maximal clique is found. Then the current v is removed from COMPSUB to NOT. The algorithm chooses a new vertex iteratively until CANDIDATES is empty (to coordinate with the description, we use same signs as [6] describes) (Figure 1a). Also there is an example shown the basic procedure for this algorithm (Figure 1b). The worst-case time complexity is proved to be $O(3^{n/3})$ [8].

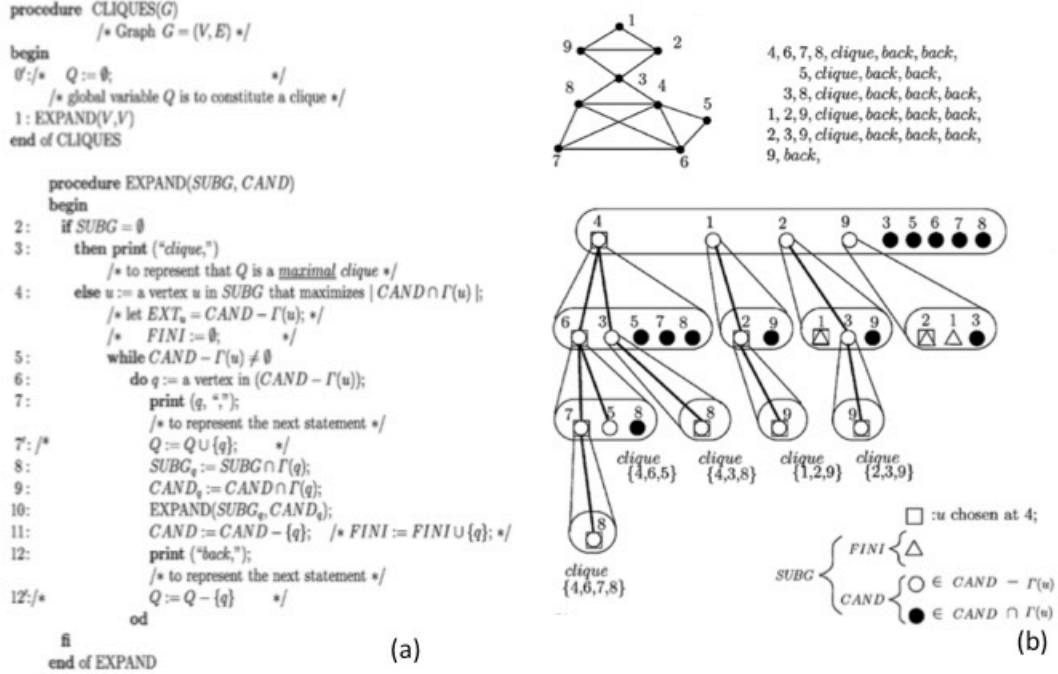


Figure 1. Bron's Algorithm. (a) The algorithm details; (b) an example (Figures are clipped from the publication [8], please refer to the publication if the figure is unclear).

Yun's Algorithm aims at genome-scale memory-intensive applications in system biology. They claim that the framework takes advantages of large-memory architectures by creating globally addressable bitmap memory indices. The basic procedure of this algorithm is as follows:

Step1: Using a maximum clique algorithm to determine upper bound on clique size.

Step2: Enumerate all k -cliques (maximal and non-maximal). In this step, they claimed that the main disadvantage of Bron's Algorithm is destroyed non-decreasing order, as they require. The improved parts in Yun's Algorithm are: 1) add comparison in COMPSUB step which will examine NEW_CANDIDATES and NEW_NOT; if they are empty, a maximal k -clique is found; if not, a non-maximal k -clique is found; 2) introduce a boundary condition to skip unnecessary step when there are less than k vertices in union of sets COMPSUB and CANDIDATES because no k -clique will be found in such condition.

Step3. Using a maximal clique enumeration algorithm using non-maximal k -cliques as input. In this step, a framework called Clique Enumerator is employed, which is used to enumerate maximal cliques of size bounded by a given range, or the upper bound found in step1. This step makes use of one characteristic property of cliques, which is any k -clique ($k \geq 3$) is comprised of two $(k-1)$ cliques that share $(k-2)$ vertices, shown in Figure 2a. It implies that any $(k-1)$ clique is maximal if it is not a component of any k -cliques. They implement the algorithm to prevent repetitive generation of non-maximal clique components and reduce the storage of all k cliques and $(k+1)$ -cliques that need lots of memory. As the algorithm schema shown in Figure 2c, the algorithm treats a k -clique as a candidate to generate $(k+1)$ -cliques

and decide whether it is maximal; then determine if a k -clique shares $(k-1)$ vertices with any other k -cliques; if no common neighbors exist, the clique is maximal and is recorded; otherwise, $(k+1)$ -clique will be formed for further check. This procedure is innovatively decided by forming bit string representing common neighbors (Figure 2b). A sample graph is executed by this algorithm in Figure 2d, showing the whole procedure to list all maximal cliques. The author claims that the running time of step k is $O((n-k)^2 * N[k] + n * M[k])$ in sequential type, and the algorithm could be improved also in parallelism.

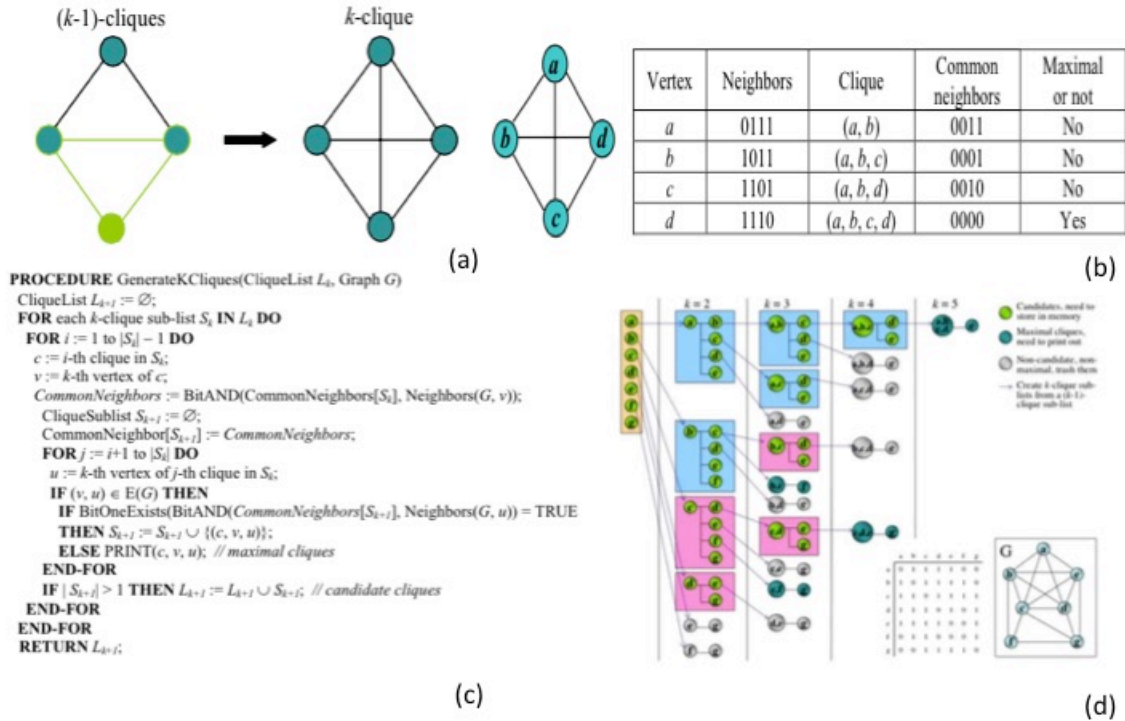


Figure 2. Yun's Algorithm. (a) A k -clique; (b) An example of bit strings representing common neighbors; (c) Pseudocode of the algorithm; (d) example of the algorithm running in a graph. (Figures are clipped from the publication [6], please refer to the publication if the figure is unclear)

Goals:

In this part, we aim to 1) implement two algorithms shown above and validate the correctness; 2) compare performance of two algorithms and investigate the effect of DOULION algorithm on k -clique problem algorithms.

Implementation:

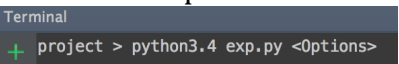
The implementation of two algorithms are included in KCLIQUE.py and exp.py written in Python3.4. KCLIQUE.py includes both algorithm detailed implementations

and exp.py includes operations and I/O for algorithms. In the implementation, we utilize a python package called networkx. The package, as the official claims, is for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [9]. For our implementation, we mainly use networkx to build and maintain the graph. Both implementations are in sequential pattern. We didn't implement the parallel pattern, although one of their advantages is parallelism; the reason is sequential algorithms could be clearer to clarify what effect DOULION algorithm will have on the procedure of these two algorithms.

Yun's Algorithm is implemented in class KCliques_Yun. In the method maximak_cliques_sequential(), we initialize sublists first by self.init_sublists(), and use a queue to pop and push each sublist in sublists. when it is popped, common neighbors of all its nodes will be checked in method clique_neighbors() and determine the node storing destination to result list or sublists as the algorithm describes; note that in clique_neighbors(), every neighbor will be checked only when it has higher index than original node to maintain non-descending order as this algorithm's property; also in original algorithm, a binary string will be build to check common neighbors; here we use networkx's neighbors() method to realize that. When calling Yun's Algorithm, maximum clique number and all maximal cliques whose k ranges from 3 to maximum number will be generated at the same time. Thus this implementation could examine all maximal cliques listing and maximum cliques problem at the same time. Also there is a critical difference of Yun's Algorithm from brute force algorithm: that is this algorithm could be generate results in a specific k value; instead it will always generate all cliques within all k values in one running round.

Bron's Algorithm is implemented in class KCliques_Bron. This implementation follows the similar implement structure as Yun's Algorithm implementation did; while its core part will make use of a function called find_cliques(). This method is modified to control k clique range in method maximal_cliques(). Note that this algorithm can run out of memory for large graphs; thus for experiments, proper graphs should be carefully chosen or constructed.

Graph Generation and Experiment operations are set in file exp.py. Class GenerateGraph is used to generate different types of graph; method runExp is designed to run a single round experiment, while method runExps is used to run several rounds experiments. We control the input parameter through terminal in

Mac: . Several options are available. Note that at least one option of input should be given in Table1. Input file format should follow the format in raw_data.txt provided.

Short	Full	Type	Default	Function
-f	--file	string	None	The input file
-d	--draw	int	0	0 not to draw result; 1 to draw result

-s	--sample	string	None	Sample graph;
				options include “karate_club” or “random_graph”
-n	--nodes	int	None	Set node number; must be set when --sample is set “random_graph”
-m	--edges	int	None	Set edges number; must be set when --nodes is set
-g	--degree	float	None	Set degree of graph range (0,1); must be set when --nodes is set while --degree is not set
-r	--shrink	float	1	Shrink input ratio.
-t	--times	int	1	Number of rounds experiments
-e	--threshold	float	inf	Upper bound k number to terminal the algorithm
-a	--algorithm	string	bron	Algorithm type; options include “bron” and “yun”

Table 1. Command Options.

Experiments and Results:

Stage 1: Validation of Two Implementations.

In this stage, we will use several basic arbitrary graphs to validate the correctness of two algorithm implementation. Because both implementations are deterministic, the correctness of implementations will not be related to structure of graph; once we validate correctness in small graphs, the implementation should work correctly in other types of graphs. We manually calculate the properties of the sample graphs first, then compare results generated from two implementations. The topological structure of four different types of graphs generated by Yun’s Algorithm and Bron’s Algorithm are shown in Figure 3 and Figure 4. We compared the node number, edge number, maximum k value, number of cliques with k from 3 to Maximum shown in Table 2. Note that for random graph, it is different in every experiment, thus manual calculation will be processed after graph is generated by the implementation with the reason that both implementations don’t change graph structure. No difference is found, thus the correctness of two implementations are confirmed. Although manual verification is tedious and inefficient, it makes sense for tiny graphs with little bias.

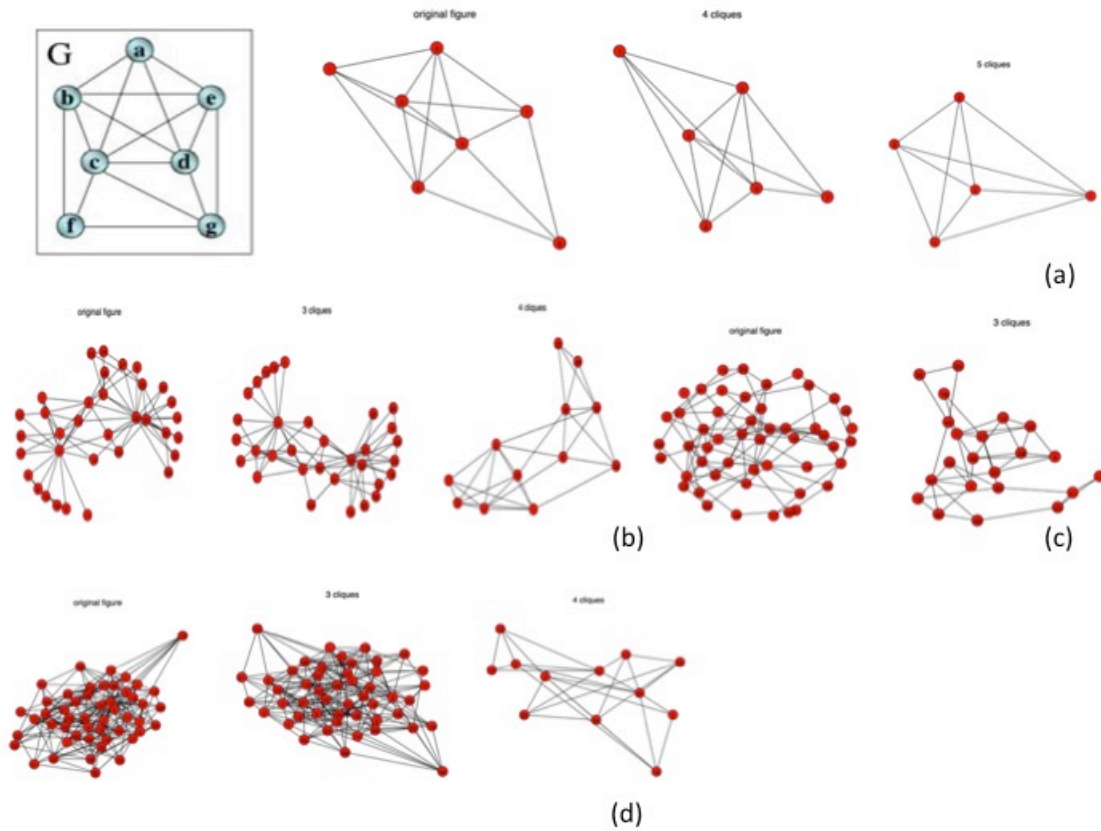


Figure 3. Sample graphs provided to validate the correctness of Yun's Algorithm. Here shows the topological structure of each type of graphs with different k-cliques included. (a) Graph from Figure 4 in publication [4]; (b) Zachary's Karate Club from package networkx; (c) Random Graph with nodes=50 and edges=100; (d) Random Graph with nodes=50 and degree=0.1.

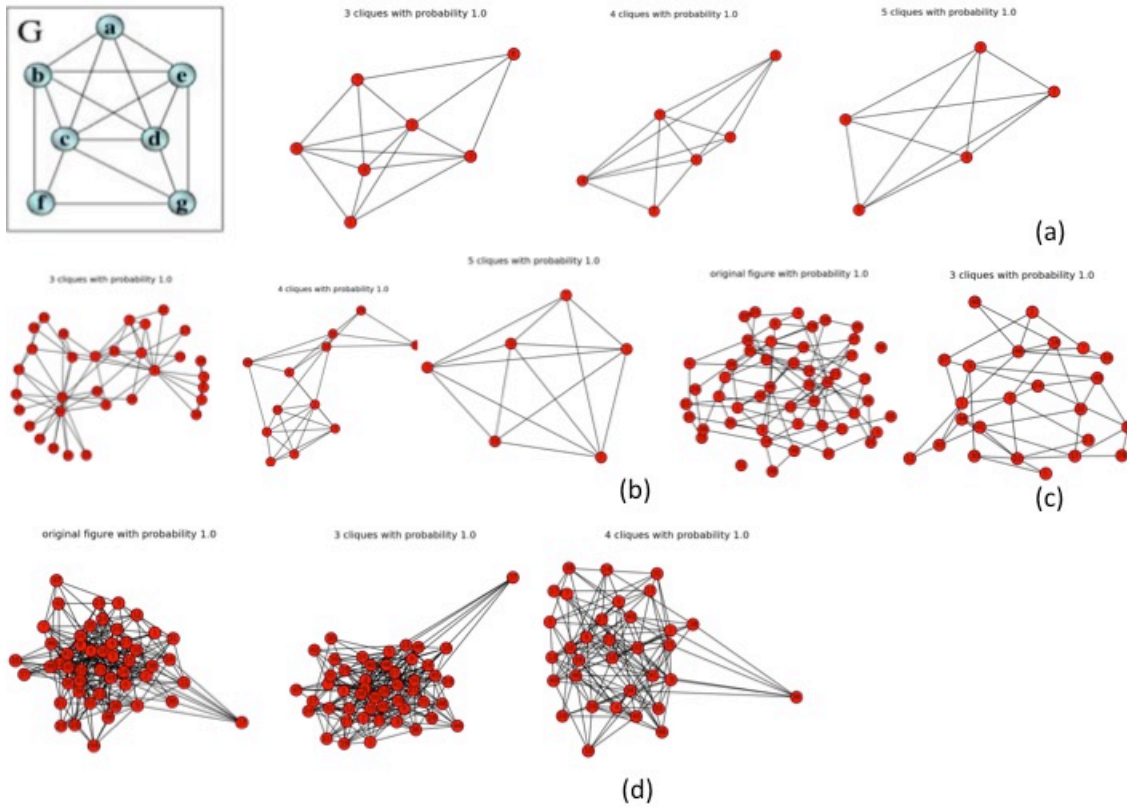


Figure 4. Sample graphs provided to validate the correctness of Bron's Algorithm. Here shows the topological structure of each type of graphs with different k-cliques included. (a) Graph from Figure 4 in publication [4]; (b) Zachary's Karate Club from package networkx; (c) Random Graph with nodes=50 and edges=100; (d) Random Graph with nodes=50 and degree=0.1.

Figure4	Nodes	Edges	Maximum k	Clique Number
Manual	7	16	5	[15,6,1]
Yun's Alg.	7	16	5	[15,6,1]
Bron's Alg	7	16	5	[15,6,1]
KarateClub	Nodes	Edges	Maximum k	Clique Number
Manual	34	78	5	[45,11,2]
Yun's Alg.	34	78	5	[45,11,2]
Bron's Alg	34	78	5	[45,11,2]
Random type1	Nodes	Edges	Maximum k	Clique Number
Manual	50	100	3	[13]
Yun's Alg.	50	100	3	[13]
Manual	50	100	3	[14]
Bron's Alg	50	100	3	[14]

Random type2	Nodes	Edges	Maximum k	Clique Number
Manual	50	226	4	[112,3]
Yun's Alg.	50	226	4	[112,3]
Manual	50	272	4	[187,21]
Bron's Alg	50	272	4	[187,21]

Table 2. Table for graph information generated in different method. Not that in Clique Number column, the list means the number of cliques with k in range with 3 to maximum k value.

Stage 2: Comparison of Two Implementations.

Under the result that two algorithm implementations' correctness are confirmed, we need to compare the performance of them and choose one of them for further research. The reason to do that is because both implementations are deterministic; to investigate the effect of DOULION algorithm on k-clique counting problem, choosing better implementation could increase research efficiency. The graphs we chose is Karate_club graph and Friends connection data from Weiliang Xing's facebook account. The graph information shown in Table 3, is generated from two implementations with same result. Note that the graph information is generated when sampling probability equals one, which meaning there is no DOULION algorithm working in the whole procedure. As DOULION mechanisms explained in [7], to set a random variable for weight assignment for each chosen edge, the algorithm randomly tosses coin biasedly, which means probability for tails and heads are not equal. This means that sampling probability may varies. In our experiment setting, we will investigate the effect of DOULION on k-clique counting algorithms at each probability ranging from 0.1 to 1.0 with interval as 0.1.

	Karate_club	Weiliang'sFB
Nodes	34	144
Edges	78	802
Denstiy	0.139	0.078
Cluster Coeff	0.571	0.596
Transitivity	0.256	0.263

Table 3. Basic information for graphs used for performance comparison for two algorithm implementations.

First we compare the time cost for both graphs run by two implementation at different sampling probabilities. Time cost means how much time needed to finish for all probabilities in one graph in one round for the specific algorithm implementation. The results are shown in Figure 5. The results shows that Bron's Algorithm implementation shows a stable time cost over different sampling possibilities in the same graph, while Yun's Algorithm implementation running time

increases as sampling possibilities increases. For example, Bron’s implementation’s running time stabilizes within 2 milliseconds in karate_club and 8 milliseconds in Weiliang’s Facebook graph, while Yun’s Algorithm’s time cost performance is worse. Thus, this experiment shows that Bron’s Algorithm implementation has better performance on time cost than Yun’s Algorithm implementation.

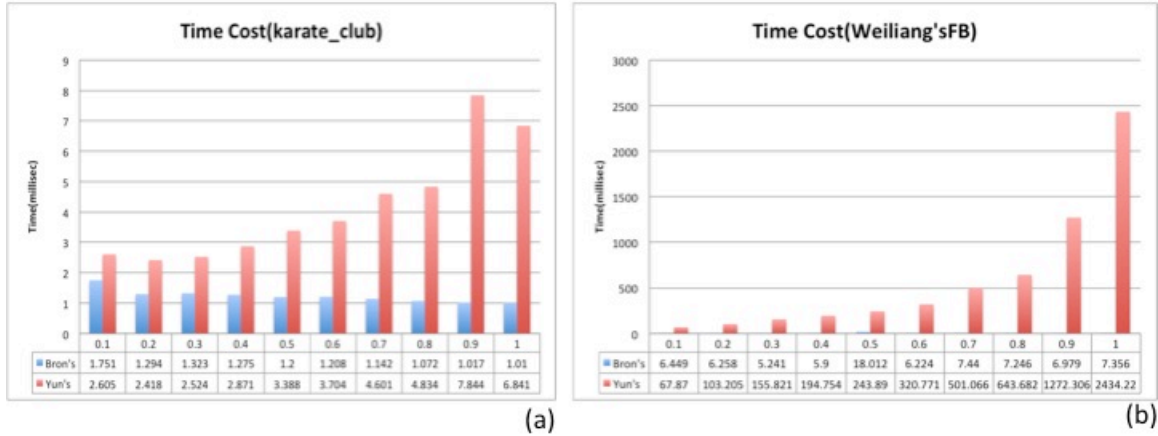


Figure 5. Time cost comparison For two algorithm implementations at different sampling probabilities. (a) Time cost for karate_club graph; and (b) Time cost for Weiliang’s Facebook Graph.

Next we compare the performance on accuracy for two algorithms implementations. Here the accuracy is defined as the percentage of number of cliques when k is at certain value at certain sampling probability over that whose sampling probability is 1.0. The result is shown in Figure 6. Theoretically, better show should be presented in table; for convenience purpose, we present here as Figure. As shown in Figure 6a and Figure 6b, each row represents two parts: number of cliques and its corresponding accuracy at certain sampling probability with the ascending order in cliques k 's value. The red frame in Figure 6b shows that Bron’s Algorithm implementation has a good accuracy estimation in Weiliang’s Facebook graph, while Yun’s estimation is not; it is noted that neither of two implementation performs accuracy well in karate_club graph, with the possible reason that the graph is too tiny, which will greatly influence sampling estimate accuracy with little statistical reliability.

Combined results from running time comparison and accuracy comparison, we tend to believe that Bron’s Algorithm implementation has better performance over Yun’s Algorithm implementation. Thus in below experiments, we will use Bron’s Algorithm implementation as main k -clique counting implementation, with DOULION’s Algorithm built on it to investigate the effects.

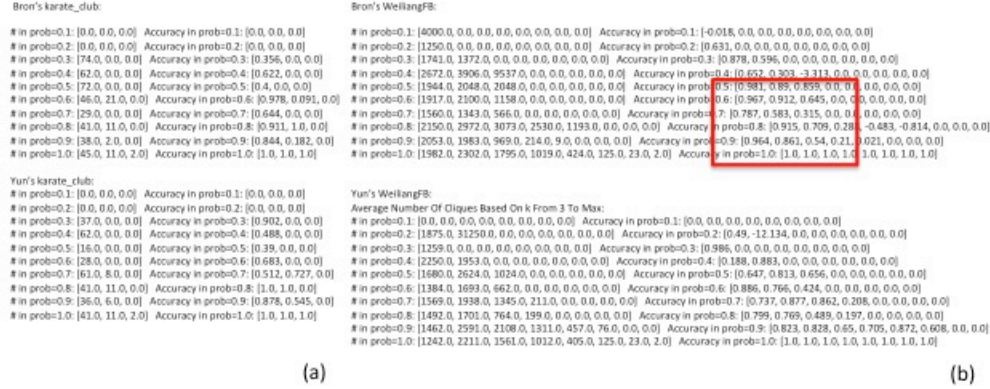


Figure 6. The accuracy performance for two algorithm implementations in a) Karate_club; and b) Weiliang's Facebook graph.

Stage 3: Experiments and Results with various types of graphs.

The first experiment is processed on Karate's Club graph with properties shown in Figure 7 with only one round. Here "one round" means we process the graph with Bron's Algorithm implementation for only one running round with results for counting all k-cliques; thus, repeated experiments could be done in many rounds define with commands shown in Table 1. It is meaningful to compare the effect on 1 round and several different rounds with the reason that DOULION algorithm is essentially a random sampling algorithm; thus different level of repeating rounds may or may not have significant effect on the results. The performance for this graph in one round is shown in Figure 8. For clique number in each probability, the number for 3-5 cliques cannot be detected when probability is less than 0.7; also there is an obvious outlier for 3 cliques when probability is 0.1. It is reasonable because for tiny graph like Karate's Club graph, DOULION randomly sampling could hugely effect/break the original graph structure which may bring in unexpected results. The accuracy shown in Figure 7 fluctuates hugely in low sampling probabilities; while they quickly approach to the referenced number when probability is 1.0 (which is regarded as most accurate) as probability increases; with the reason that as sampling probability increase, edges of graphs have more chanced to be selected, which will have less effect on graph's structure, resulting in more accurate result. The speedup is unexpected in one round experiment. Because theoretically in lower probability level, less edges are selected and more speedup is achieved; however in Figure 7 bottom figure shows the unobvious speedup. Due to the potential instability of random sampling and tiny graph scale, speedup might be improved over repeated experiments.

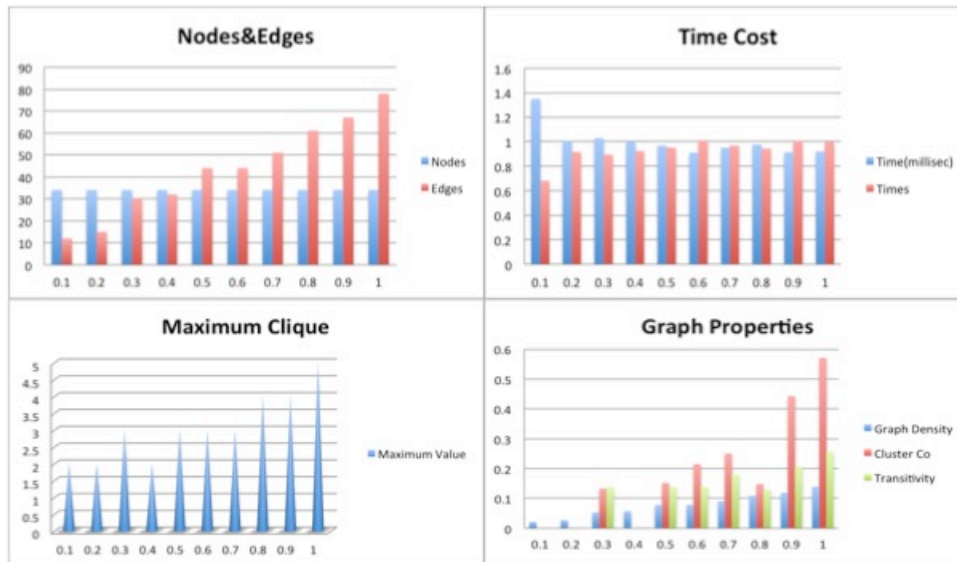


Figure 7. Property of Zachary's Karate Club generated in one round.

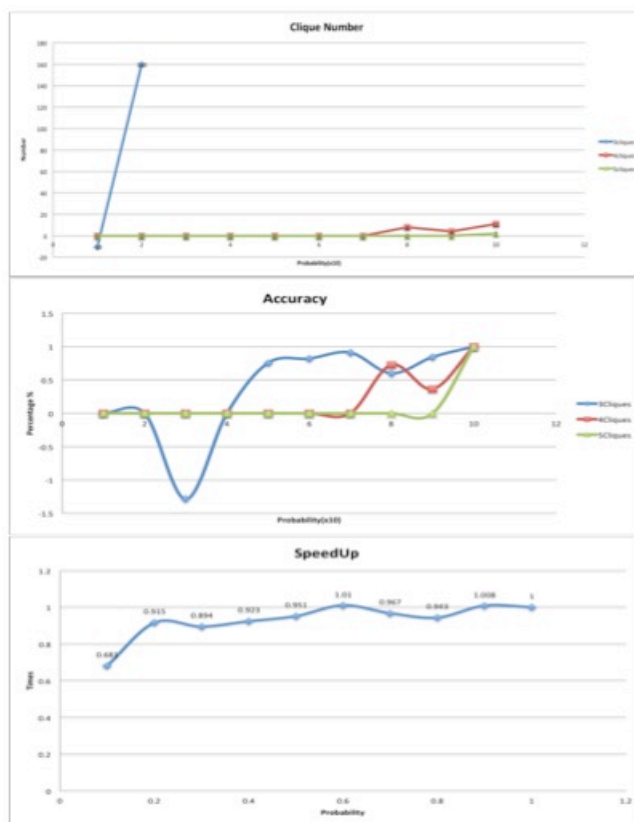


Figure 8. Performance of Bron's Algorithm with DOULION Algorithm from 3 cliques to 5 cliques.

For Karate Club Graph, repeated experiments are processed in this part. The results are averaged from all repeated rounds' results and drawn in Figure 9. The effect on graph's property generated by repeated experiments has few difference with on

round experiments. The results are shown in Figure 9. For maximum clique counting, the results are similar, indicating that repeated experiments did not affect maximum clique generation; for accuracy comparison, the shapes of each experiment are different; however, for all of the figures, 3-clique counting (or triangle counting) shown in blue curve in second column of Figure 9 could quickly approaches 1.0 in each level of sampling probability except 0.1, while 4-clique counting and 5-clique counting is less stable even we increase the repeated number; For speedup, it is clear that the statistics become stable and reasonable as repeated rounds increase, with less and less outliers appear; also speedup is higher in low sampling probability than high sampling probability in 50 rounds shown in Figure 9, which accommodates with the expected results.

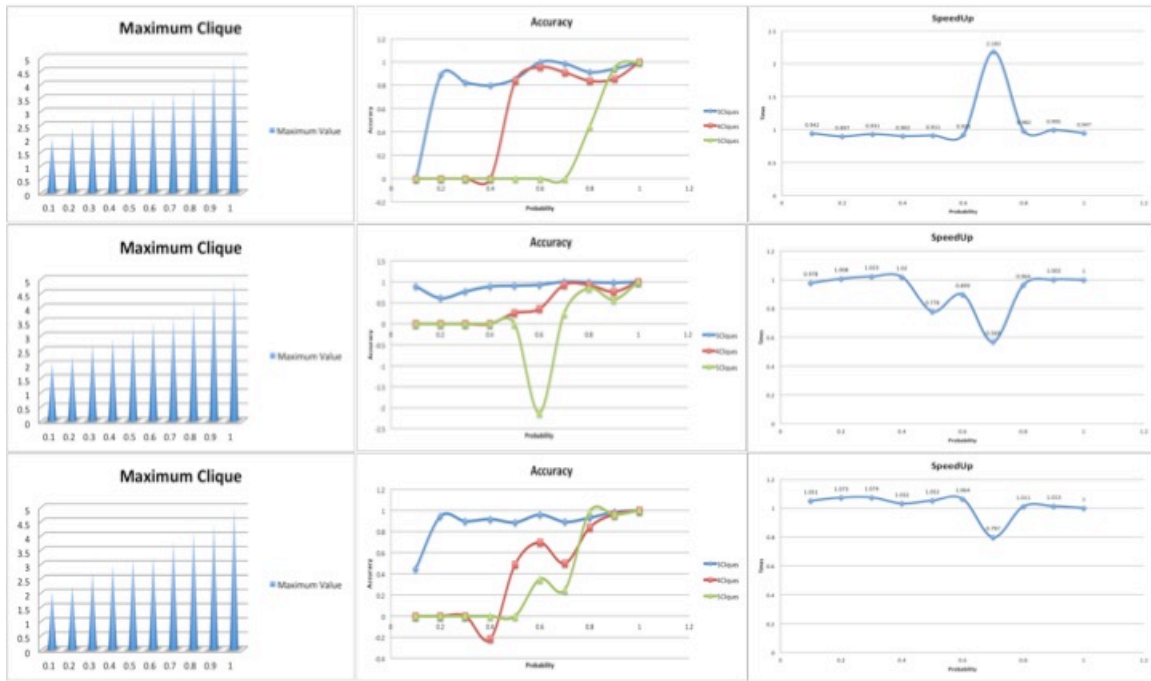


Figure 9. Performance Repeated experiments with different rounds for Karate's Club graph. Top row is 10 rounds; middle row is 20 rounds; and bottom row is 50 rows.

We did the similar experiments in different graphs. Next target graph is Weiliang's Facebook graph, with the performance results shown in Figure 10. It is clear also that repeated experiments have little effect on maximum clique generation; also the result that the value of maximum cliques generation decreases as sampling probability decreases. It is reasonable because for maximum clique, its distribution is relatively sparse, which is very difficult for lower sampling probability to "hit" edge of the maximum clique, and this will result in decreasing maximum clique value generation. Speedup is greatly improved and stable after reasonable number of repeated experiments, as shown in the second column in Figure 10. We compare the accuracy based on different probabilities and based on different cliques in Figure 10. It clearly shows that as the repeated rounds increase, the property of

accuracy for each cliques in each probabilities stabilized. In lower clique problem, for example 3-cliques problem, the accuracy quickly approaches 1.0 in low sampling probability (bottom figure in third column of Figure 10); while in higher cliques, number of cliques is difficult to be detected in low sampling probability. This result coordinates with maximum cliques generation we discussed earlier.

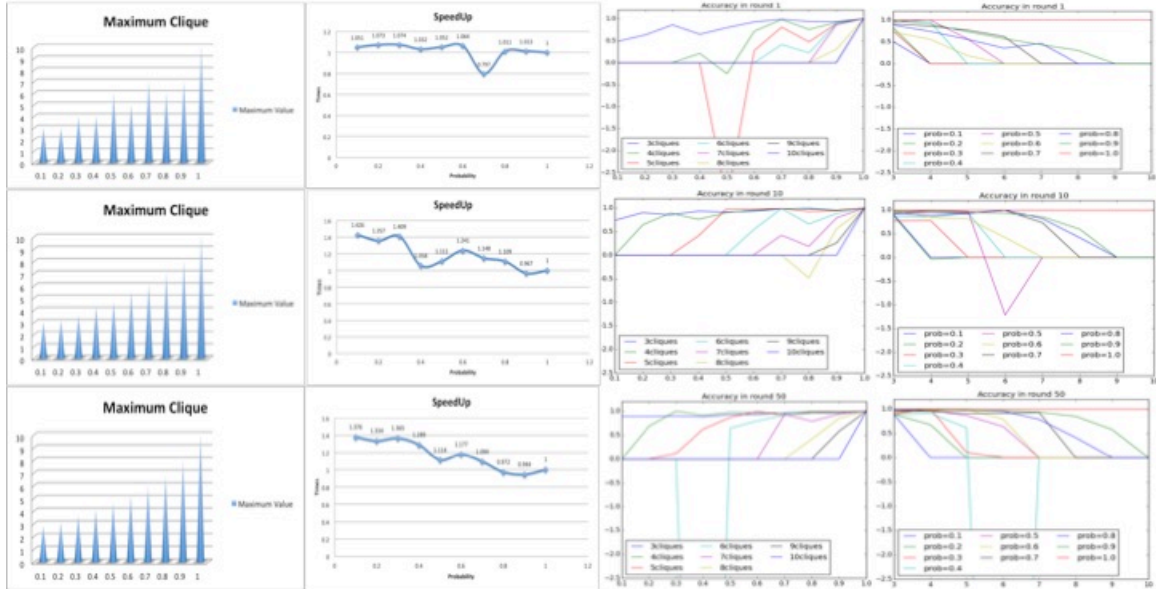


Figure 10. Performance Repeated experiments with different rounds for Weiliang's Facebook graph. Top row is 1 round; middle row is 10 rounds; and bottom row is 50 rounds. The first column is the maximum clique counting in each sampling probability; the second column is the speedup in each sampling probability; the third column is the accuracy comparison with different cliques in each sampling probability; and the fourth column is the same accuracy comparison with different sampling probability in each different clique.

Next experiment's target is Jian's Weibo Graph which relatively large. The similar experiment is performed and the performance results are shown in Figure 11. For larger graph, repeated experiments show more obvious effect on speedup results, which makes curve more smooth as shown in Figure 11. It also shows stabilized effect on accuracy when repeated rounds increase which is same as previous discussed.

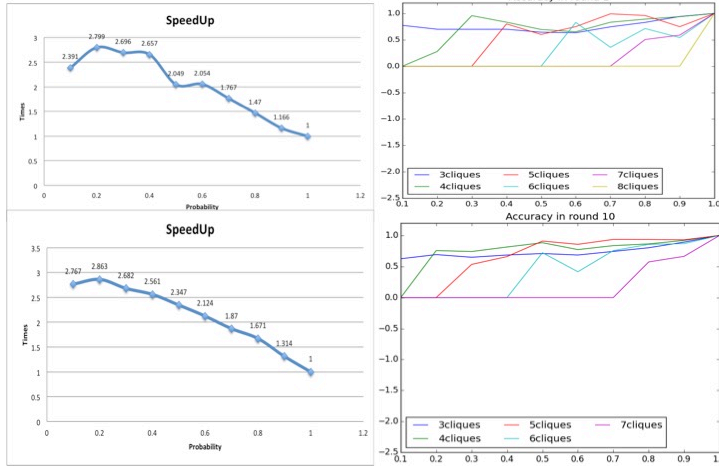


Figure 11. Performance Repeated experiments with different rounds for Jian’s Weibo graph. Top row is 1 round; and bottom row is 10 rounds. The first column is the maximum clique counting in each sampling probability; the second column is the accuracy comparison with different cliques in each sampling probability.

Discussion and Conclusion:

DOULION algorithm is originally invented and investigated for triangle counting problem which shows good properties in accuracy and speedup time [7]. Our study here tries to extend the application of DOULION algorithm to general k -clique problem. In part II, we first implemented two algorithms called Yun’s Algorithm and Bron’s Algorithm which are named by their first author [4][6]. After confirming the correctness of two implementations and comparing performances over speedup and accuracy through experiments, Bron’s Algorithm implementation is chosen for further study. We then combine DOULION algorithm with Bron’s Algorithm, and compare the performance for different k -cliques, different sampling probabilities, and different repeating times for various graphs with different structures. Experiments show that DOULION algorithm could improve speedup time for general graphs; however, things become complex when involving in accuracy qualities. When meeting with small graphs like Karate’s Club’s graph, accuracy is unstable. DOULION algorithm is essentially a random sampling algorithm. For low sampling probability, high k -clique counting, or small graphs, it could not avoid unexpected results such as outliers in the resulting graphs; this is also true for other graphs, when running only one round, curves that represent accuracy at different sampling probabilities keep fluctuating, which made results less reliable; however, this effect could be relieved when meeting with large graphs with low k -clique counting; for example, triangle counting would perform well at even low sampling probabilities with no need to do repeated experiments for large graph; however, for higher k -clique counting, repeated experiments could smoothing the accuracy fluctuation with enough repeating rounds. After all, for general graphs, when k increases, k -clique would more rarely appear in the graph comparing to large amount of triangles, which give less chance to let DOULION algorithm to “hit” the k -clique and estimate its number. Another improvement of repeating experiments is that it could

smooth the curve represent for speedup time based on sampling probabilities; in other words, the total performance of the algorithm would behave more stable.

In conclusion, DOULION algorithm combining with proper k-clique counting algorithm could improve speedup and accuracy quality, thus improve the performance; to enhance its reliability and stability, repeated experiments are needed for proper repeating times. This part shows the potential power of DOULION algorithm in k-clique counting algorithm which has a promising future. However a lot of future work need to be done in this area, including linking key properties of the graph with this algorithm to achieve best performance; finding maximum clique with acceptable performance using this algorithm; find proper repeating times to have acceptable results, etc. This part, through innovatively combing DOULION algorithm with k-clique counting algorithms, presents interesting and appealing prospect in graph study.

Acknowledges:

We would like to express our very great appreciation to professor Jie Gao for useful tutoring and guidance.

References:

- [1] http://en.wikipedia.org/wiki/Clique_problem.
- [2] http://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems#CITEREFKarp1972.
- [3] Moon, J. W.; Moser, L. (1965), "On cliques in graphs", *Israel Journal of Mathematics* 3: 23–28.
- [4] Bron, C.; Kerbosch, J. (1973), "Algorithm 457: finding all cliques of an undirected graph", *Communications of the ACM* 16 (9): 575–577.
- [5] Cazals, F.; Karande, C. (2008), "A note on the problem of reporting maximal cliques" (PDF), *Theoretical Computer Science* 407(1): 564–568.
- [6] Zhang, Yun, et al. "Genome-scale computational approaches to memory-intensive applications in systems biology." *Supercomputing*, 2005.
- [7] Tsourakakis, Charalampos E., et al. "Doulion: counting triangles in massive graphs with a coin." *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009.
- [8] Tomita, Etsuji, Akira Tanaka, and Haruhisa Takahashi. "The worst-case time complexity for generating all maximal cliques and computational experiments." *Theoretical Computer Science* 363.1 (2006): 28-42.
- [9] <https://networkx.github.io/>