

# **Project report-**

## **A comparison and evaluation of Minimum Cut algorithm**

Name: Qijie Chen

Student number: 250858536

Department of Computer Science

### **Introduction**

For the final project of the course Analysis of Algorithm II, I choose a topic to analyze and evaluate two of existed algorithms about the Minimum Cut searching problem. The first one is implemented by Mechtild Stoer and Frank Wagner in 1994, which will be referred as “Stoer-Wagner’s algorithm” in this paper. Another algorithm is implemented by David Karger in 1993, which will be referred as “Karger’s algorithm” in this paper. In this project. First I need to implement both algorithms in C++ programs. Then I need to prepare a series of test samples to run some tests on my algorithm implementation. Finally, I try to evaluate their performance by some of measurements like running time, approximation ratio and the results of different test cases, in the meanwhile try to find some approaches to make the algorithms better.

To do actual work for this project, first we should figure out some relative mathematic concepts in graph theory. In the graph, The cut  $C=(S,T)$  represents a partition of the vertex set  $V$  of the graph  $G=(V,E)$  into two disjoint subsets  $S$  and  $T$ , and the cut-set  $C=(S,T)$  is a edge set  $\{(u,v) \in E \mid u \in S, v \in T\}$  whose have one endpoint in  $S$  and another end point in  $T$ . Basically, The target of minimum cut algorithm is to find an effective approach to find a cut which can separate the graph vertex set  $V$  into two sets  $S$  and  $T$ , and the sum of the weights of edges connecting these two sets  $S$  and  $T$  is minimum..

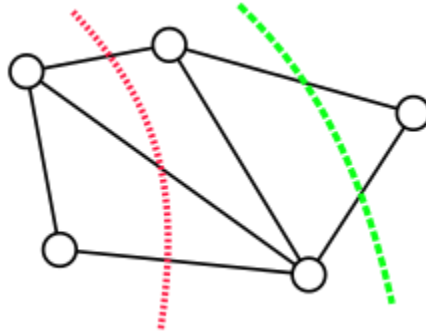


Figure 1 A classic representation of minimum cut (Two cuts represent here and the latter one is the minimum cut in this graph)

Many algorithms can solve minimum cut problems. The most common approach is transform such problems into a kind of similar problems-maximum flow problems. Maximum flow problems states a fact that in a flow network, the maximum amount of flow passing from the source vertex  $s$  to the sink vertex  $t$ , is equal to the capacity of minimum cut in the network. Hence, to find minimum cut is equal to find the maximum flow in the network. One of the most famous algorithm to solve maximum flow problems is Ford-Fulkerson method. And there is an optimal algorithm implementation based on Ford-Fulkerson method is called Edmond-Karp algorithm. The idea of Ford-Fulkerson method is very simple: as long as there is a path from source  $s$  to the sink  $t$ , and the capacity of every edge in this path is not equal to zero, such a path is called augmenting path. Then we can send flow along an augmenting path and modify the capacity of each edges in the path after flow sending. Finally we can get maximum flow of the network if there is no augmenting path existed in the network. In the original Ford-Fulkerson method, it never specify the strategy to select which augmenting path in the network. Hence, Edmond-Karp algorithm specify it should select the shortest augmenting path in each iteration of Ford-Fulkerson method, and such a breadth first search strategy can make Ford-Fulkerson method better. Although Edmonds-Karp algorithm can solve the maximum algorithm in  $O((V^2E))$  time, which is a remarkable performance, It can not be effective to solve minimum cut problems. Because in the maximum flow problem, it specify the unique source and sink in the graph. But in the minimum cut problem, since there are no fixed source and sink, which means any vertex in the graph can be considered as source or sink. If there are  $n$  vertices in the graph, we can define  $n*(n-1)$  source-sink pairs in the graph, and then we need to run  $n*(n-1)$  times of maximum flow algorithm to get the minimum flow of the graph. Even

though based on the flow equivalence tree theory introduced by Gomory and Hu, which prove that the minimum cut can be solved by  $n-1$  maximum flow problems, by defining a fixed vertex  $s$  and all  $n-1$  possible choices of  $t$  which satisfied  $\{t | t \in V - s\}$ , it still a time-consuming task .

Therefore, we turn to find better and more efficient algorithm to solve minimum cut problems here. There are two outstanding and famous minimum algorithm that worth analyzing in this project. The first one is called “Stoer-Wagner’s algorithm”, a recursive algorithm based on a simple strategy. Another algorithm is called “Karger’s algorithm”, which is a randomized algorithm. In the following pages we will introduce these two algorithm precisely.

## Algorithm description

### Stoer-Wagner’s algorithm

Stoer–Wagner algorithm is are recursive algorithm which can effectively solve the minimum cut problem in undirected weighted graph. It was first published by Mechthild Stoer and Frank Wagner in 1995. The essential idea of this algorithm is quite simple, We can keep shrinking a graph by merging two most intensive vertices until the graph only contains two vertex. And for each iteration of shrinking phase, the weight of the merged cut will be stored in a list. When the algorithm come to the end, we can compare those cuts in the list and the minimum cut in the list will be the minimum cut of the graph.

This algorithm can be divided into two phases: minimum cut phase and minimum cut finding phase. The minimum cut phase is a recursive phase to shrink the graph by merging vertices, and record weights of the merged cut. The procedure of minimum cut phase can be represented like this:

***MINIMCUT PHASE ( $G, w, a$ )***

***$A \leftarrow \{a\}$***

***While  $A \neq V$***

***Add to  $A$  the most tightly connected vertex***

***Store the cut-of-the-phase and shrink  $G$  by merging the two vertices added***

***last***

This procedure was provided by Stoer and Wagner's paper, Basically it describe a procedure like this. First, we can start with picking an arbitrary vertex  $a$  from the graph  $G$ , which is an initial element in  $A$ . Then, we need to find a vertex outside of  $A$  which most tightly connected to  $A$ , and then add this vertex to  $A$  either. Most tightly connected vertex means the edge between this vertex and  $A$  has the maximum value of weight among all the edges connected between  $A$  and the vertices outside of  $A$ . And finally, we need merge two vertices added last in the vertex set  $A$  by replacing with a new vertex. And we need to merge edges from these two vertices to a single remaining vertex outside of  $A$  into a new edge, the weight of this edge will be the sum of weights from previous edges.

And we need to notice that in each iteration of merging phase, we need to store the cut-of-the-phase., which represent a cut of  $V$  that separates the vertex added last from the rest of the graph. And the minimum cut-of-the-phase will be the desired minimum cut that we want to find through the algorithm.

By adding the minimum cut finding phase we can get the whole procedure of Stoer-Wagner's algorithm.

***MINIMUMCUT( $G, w, a$ )***

***While  $|V| > 1$***

***MINIMUMCUT PHASE ( $G, w, a$ )***

***If the cut-of-the-phase is lighter than the current minimum cut***

***Then store the cut-of-the-phase as the current minimum cut***

We should keep the same starting vertex  $a$  through the whole algorithm, but in each phase, we can select a random vertex. Here is a brief example to show how the Stoer-Wagner's algorithm working.

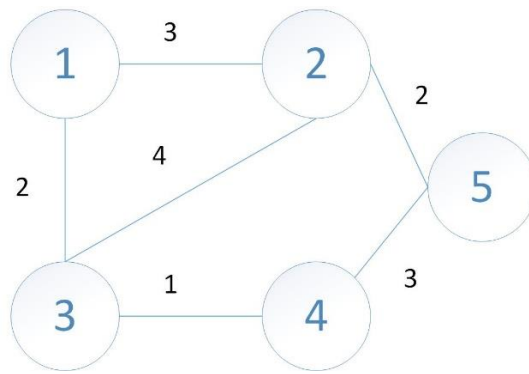


Figure 2 The original graph, we first select vertex 1 as our start point with partition  $\{1\}, \{2, 3, 4, 5\}$  of  $V$ , and the weight of cut-of-the-phase is 5

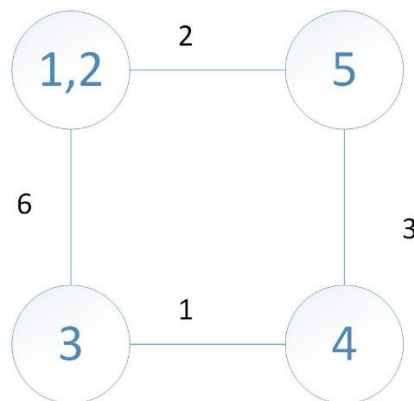


Figure 3 After first MINIMUM CUT phase  $(G, w, 1)$ , we merge vertex 1 and vertex 2 since the edge between them is most tightly connected, then update and modify rest edges. In the second MINIMUM CUT phase  $(G, w, 4)$ , we randomly select vertex 4 as a with partition  $\{4\}, \{1, 2, 3, 5\}$  of  $V$  and the weight of cut-of-the-phase is 4

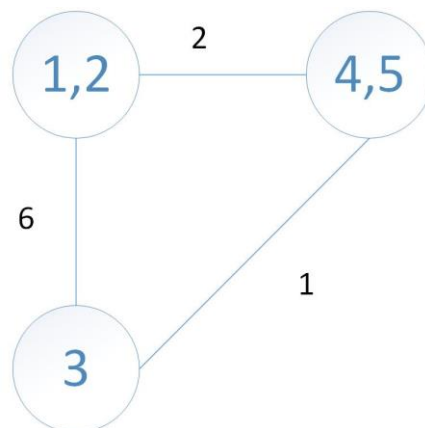


Figure 4 After second MINIMUM CUT phase  $(G, w, 4)$ , we merge vertex 4 and vertex 5 since the edge between them is most tightly connected, then update and modify rest edges. In the third MINIMUM CUT phase  $(G, w, (1, 2))$ , we randomly select vertex (1, 2) as a with partition  $\{1, 2\}, \{3, 4, 5\}$  of  $V$  and the weight of cut-of-the-phase is 8



Figure 5 After third MINIMUM CUT phase  $(G, w, (1, 2))$ , we merge vertex  $(1, 2)$  and vertex 3 since the edge between them is most tightly connected, then update and modify rest edges. In the fourth MINIMUM CUT phase  $(G, w, (1, 2, 3))$ , we randomly select vertex  $(1, 2, 3)$  as a with partition  $\{1, 2, 3\}, \{4, 5\}$  of  $V$  and the weight of cut-of-the-phase is 3. And since there are two points left in the graph  $G$ . After merging these two points, the algorithm is terminated. And we can find the minimum of cut-of-the-phase is 3, which is also the minimum cut of the graph  $G$ .

The time complexity for this algorithm is simple. We can see that we need to run  $|V|-1$  MINIMUM CUT phase to merge all vertices in the graph  $G$ . And for single MINIMUM CUT phase, we need  $O(|E|)$  time for determine the most tightly connected edge and compute the cut-of-the-phase, and need  $O(|V| \log |V|)$  to update and modify edges from the influence of merging. Hence we need  $O(|E| + |V| \log |V|)$  time for MINIMUM CUT phase. And the time complexity of the whole algorithm will be  $O(|V| |E| + |V|^2 \log |V|)$ .

## Karger's algorithm

Karger's algorithm is another effective minimum cut algorithm based on a randomized algorithm, which is published by David Karger in 1993. Such an algorithm use an even simpler essential idea: Doing contraction of randomly picked edge  $(u, v)$  in an undirected graph  $G = (V, E)$  by merging the vertices  $u$  and  $v$  into one vertex. And all other edges connected to either  $u$  or  $v$  will replace their endpoint  $u$  or  $v$  by the new merged vertex. This method can reserve those edges and make such edges reachable by the new vertex. Similarly, we repeat such procedure until two vertices left in the graph.

Because this algorithm is based on randomized algorithm, which can not give us a certain minimum cut of the graph from its result. Hence, for Karger's algorithm, we can not only focus on the procedures of the algorithm, but also consider its time complexity and the probability of getting a correct answer. Besides, it should be some methods to decrease its probability of failure into an acceptable level. First, it's a standard procedures of Karger's algorithm

**Procedure Contract( $G=(V,E)$ ):**

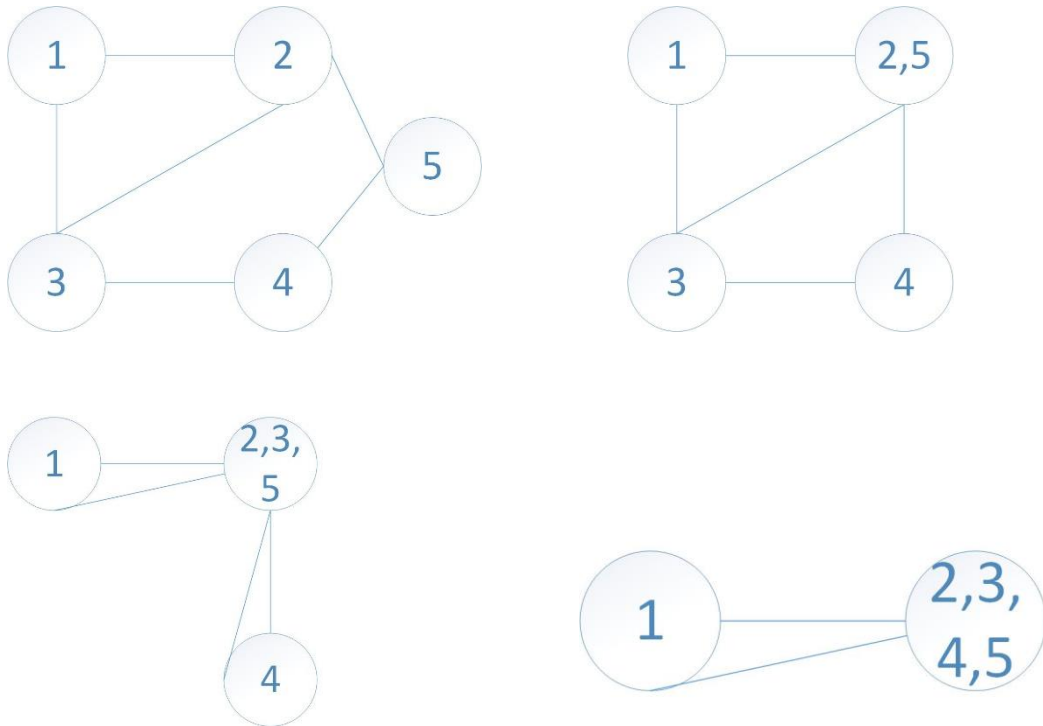
**While  $|V| > 2$**

**Choose  $e \in E$  uniformly at random**

**$G \leftarrow G/e$**

**Return the only cut in  $G$**

And here is a brief example to illustrate how the Karger's algorithm works.



From the procedures what upper graph shows, first we randomly pick the edge (2, 5) with probability of  $1/6$ . Then merged these two points into a new point and all edges connected to vertex 2 and vertex 5 originally will reschedule their paths and connect to the new vertex (2, 5). Similarly, we simulate such a random pick process and merge edge ((2, 5), 3), ((2, 3, 5), 4) in sequential order. And finally we can get two vertices remain in the graph, with two edges connected to them. So from the algorithm we get the solution is 2, which is exactly the minimum cut of our graph. However, when we apply this algorithm in a complicated graph, it may not return a correct minimum cut by a single Karger's algorithm iteration, which we will discuss later. For the convenience of understanding, we use a unweighted graph as our example. In fact, Karger's algorithm can solve minimum cut problems of weighted graph as well.

The time complexity of this algorithm is depending on how to construct the graph in our program and which strategy we use to simulate the contraction in the data structure in our program. One of typical implementation of this algorithm is using adjacency matrix or

adjacency list to represent the graph, and finish each edge contraction by updating the data structure we use to store the graph linearly, and total time complexity will be  $O(n^2)$ , where  $n$  is the number of vertex. However, there is another implementation which can give us better performance in time complexity. One implementation is using Kruskal's algorithm. We can randomly assign rank for each edge in the graph so we can use Kruskal's algorithm to construct a minimum spanning tree based on the ranks. If we remove the heaviest edge in the minimum spanning tree, the two components that can describe a cut. In this way, we can do contraction procedure like Kruskal's algorithm in a time complexity of  $O(|m| \log |m|)$  where  $m$  is the number of edges in the graph, and we need  $O(|V|)$  space to store all information of the graph. Besides, we need to focus the probability of getting a correct minimum cut by a single Karger's algorithm iteration. We assume that in the graph  $G = (V, E)$  and there are totally, and there are  $n$  vertices in the graph. It is easy to prove the probability that we can find a minimum cut for a single Karger iteration is  $(\frac{2}{n(n-1)})$ , which can be proved by following process:

We let  $C$  denote the edges of a specific minimum cut of size  $k$ , and the Karger's algorithm returns  $C$  only if none of random edges it contract through the algorithm is belong to  $C$ . For first contraction, it is easy to prove that the first edge contraction avoid  $C$  with probability  $(1 - \frac{k}{|E|})$ , the minimum degree of  $G$  is at least  $k$ , so the number of edge  $|E| > \frac{k \times n}{2}$ , Thus we can get the probability  $p_1$  that algorithm doesn't pick the edge in  $C$  in the first contraction:

$$p_1 = 1 - \frac{k}{|E|} \geq 1 - \frac{k}{\frac{kn}{2}} = \frac{n-2}{n}$$

Since only two vertices left at the end of the algorithm,  $n-2$  iteration of contraction should be performed on the Karger's algorithm. Since in each iteration  $i$ , there are  $n-i+1$  vertices left in the graph  $G$ , total number of edges left in  $G$  at least  $k \cdot (n-i+1)/2$ . The probability  $p_n$  that edges in  $C$  never pick in the  $n$ th iteration of contraction should be satisfied

$$p_n \geq 1 - \frac{k}{\frac{k(n-i+1)}{2}} = \frac{n-i-1}{n-i+1}$$

And we can conclude the probability that a single Karger's algorithm process can return the minimum cut is the product of  $p_1, p_2, \dots, p_{n-2}$ , which means in every iteration, none of edges in  $C$  is contracted



$$p = p_1 \times p_2 \times p_3 \times \dots \times p_{n-2} \geq \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \dots \times \frac{1}{3} = \frac{2}{n(n-1)}$$

We can see that it is hard to find minimum cut for single Karger's algorithm process. However, the probability can be improved if we run several times of Karger's algorithm. For example, if we run  $T = \frac{2}{n(n-1)} \ln n$  times, and the probability of not finding the minimum cut will be

$$\left(1 - \frac{2}{n(n-1)}\right)^T \leq \frac{1}{e^{\ln n}} = \frac{1}{n}$$

Which means the probability of finding a minimum cut is  $\frac{n-1}{n}$ , which is excellent for such a randomized algorithm

## Implementation

In previous sections, we take long pages to describe two algorithms we need to use in this project. In this section, I will describe how I implement these two algorithms. Experiment platform is Windows 8 system, and support software is Microsoft Visual Studio 2015. And all two algorithms I implemented in C++ version.

### Test case

For this experiment, I need to prepare several test cases to verify the performance of my implementation. However, it is not easy to prepare huge and complicated test case for the algorithm related to graph. First, it requires us to convert a graph from a visual representation to a data representation manually. It needs lots of time to observe the graph and convert carefully. Then, it is not easy to find such complicated test cases about the graph from Internet with free permission. Finally, all test cases should be calculated carefully to obtain the value of minimum cut as the referenced value, which is used to evaluate the performance of our implementation. It takes lots of time to.

For this experiment, I prepared five test cases, three of them are weighted undirected graphs. The small one contains 4 vertices, the medium one contains 8 vertices which proposed on Stoer and Wagner's paper, and the larger one contains around 20 vertices. Other two test

cases are special test cases. One of them is complete graph and each vertex has same weight which is 1, another one is unweighted undirected graph. Here is some visual components of our test cases.

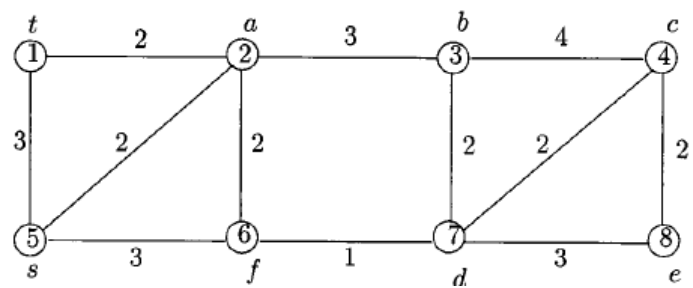


Figure 6 Medium weighted undirected graph test case

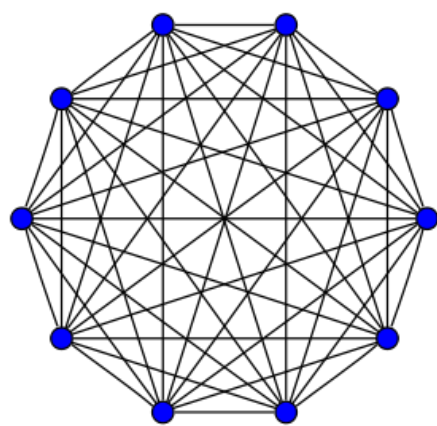


Figure 7 10-vertices complete graph test case

## Input-output Format

Input file is txt file, which contains all data which denoted from original graphs of test cases. Each row of input file record all information of a single vertex, include all its adjacent vertices and weight of edges connected between these vertices and their adjacent vertices. Here I use a picture to illustrate the input file format.

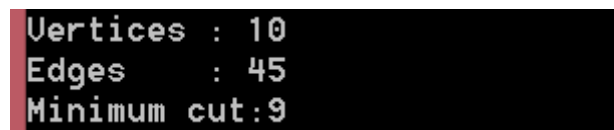
1	2	2	3	2	4	7
2	1	2	3	3		
3	1	2	2	3		
4	1	7				

Figure 8 Input file sample

This is the input file of small test case in our experiment. There are three rows in this test file, which means there are four vertices in this test case. And the first column represents target vertex; second column represents the first adjacent vertex of target vertex; third column

represents weight of edge that connected between target vertex and first adjacent vertex; fourth column represents the second adjacent vertex of target vertex; fifth column represents weight of edge that connected between target vertex and second adjacent vertex.....and so on. For example, we can see from first row, target vertex is vertex 1, and its adjacent vertices include vertex 2, vertex 3 and vertex 4, and the weight of those edges connected between vertex 1 and its adjacent vertices like edge(1,2)=2, edge(1,3)=2 and edge(1,4)=7.

Output file of Stoer-Wagner's algorithm include the number of vertices and edges in the test case. Besides, it also include the value of minimum cut of the graph defined by the test case.



```
Vertices : 10
Edges    : 45
Minimum cut:9
```

Figure 9 output screen of Stoer-Wagner's algorithm

While for the Karger's algorithm, it output the repeat times of Karger's algorithm as well. For the discussion in the previous section, we understands that we need to repeat Karger's algorithm for  $T = \frac{2}{n(n-1)} \ln n$  times to get a reliable result of minimum cut with a higher and acceptable probability. And in order to evaluate the running time of these two algorithm, the output screen will include time information in the output screen at the testing stage.

## Details in implementation

First, for both implementation, the use file screen to read all data of test case file into memory. Since we define the input file format before, and each data element is separated by a single Tab space, it is easy to operate and allocate these elements into different data structure. Besides, I maintain a global adjacent matrix for each of implementation. Adjacent matrix will be implemented by two-dimension array, and such data structure can perfectly represent vertices, adjacent vertices, edges and weight of each edge. This adjacent matrix is mainly used for data update since both algorithm have lots of merging operation, which easy to handle adjacent matrix to finish those updating operation.

For the implementation of Stoer-Wagner's algorithm. I divide the implementation into

search phase and information update information. In the search phase I use linear search to find two vertices  $s$  and  $t$  which owns a edge with maximum weight out of all edges. Then we need to merge these two point and compute the cut that separates  $t$  from the graph as the cut-of-the-phase. Finally compare the value of cut-of-the-phase with a global minimum cut value. If the value of cut-of-the-phase is smaller, than update cut-of-the-phase into global minimum cut value. And in the information update phase, we need to update the adjacent matrix since some vertices merged, and the weight of edges related to merged points require to be updated in this phase. If there are  $n$  vertices in the graph, for Stoer-Wagner's algorithms, we need to do  $n-1$  iterations which including search phase and information update phase.

For the implementation of Wagner algorithm, I define three classes to finish major operation. First one is graph class, which provide some public functions to handle the vertices and edges of graph. It also provide an adjacent list to store those vertices and adjacent vertices. Second class is called Unionsearch class, which provide some functions to handle edge merging operation and data update operation after merging. The last class is called Minimumcut class, which used to implement the major process of Wagner algorithm by calling the functions provided by other two classes, and return the value of cut after running several times of Wagner algorithm.

## Result Analysis

On this stage, we need to evaluate the performance of both algorithm implementation to determine their performances towards different test cases. The major method is using timestamp function to record start time and finish time of the whole programs or specific function. And then we can get the running time by mining the timestamp of start time and the timestamp of finish time. First, we analyze the performance of small weighted undirected graph test case, with four vertices and four edges.

Algorithm	Test 1	Test 2	Test 3	Overall result(us)
Stoer-Wagner's algorithm	583	403	492	493

Karger's algorithm	3382	2777	3454	3204(32 repeats)
--------------------	------	------	------	------------------

Comparing the results, we find that Karger's algorithm uses much more time. However, based on our implementation, Karger's algorithm repeat 32 times here, actually average running time for single Karger's algorithm is about 100 us, which only costs 20% of running time of Stoer-Wagner's algorithm.

Then we analyze the performance of medium weighted undirected graph test case, with eight vertices and twelve edges

Algorithm	Test 1	Test 2	Test 3	Overall result(us)
Stoer-Wagner's algorithm	402	561	403	455
Karger's algorithm	32044	32331	32830	32402(192 repeats)

We surprisingly find that the running time of Stoer-Wagner's algorithm almost remains the same as the previous test! While the running time of Karger's algorithm increase 10 times. But since there are 192 repeat here, actually running time of single Karger's algorithm is 169 us, 60% increase comparing to the previous test.

Then, I post the test result table for large weighted undirected graph test case, complete graph test case and unweighted undirected graph test cases here.

Test result for large unweighted undirected test case (32 edges and 21 vertices)				
Algorithm	Test 1	Test 2	Test 3	Overall result(us)
Stoer-Wagner's algorithm	547	464	580	530
Karger's algorithm	1899502	1892404	1891954	1894620(1764 repeats)

Test result for complete graph test case (45 edges and 10 vertices)				
Algorithm	Test 1	Test 2	Test 3	Overall result(us)
Stoer-Wagner's algorithm	463	454	483	467
Karger's	129303	127297	127543	128046(300 repeats)

algorithm				repeats)
-----------	--	--	--	----------

Test result for unweighted undirected graph test case (13 edges and 10 vertices)				
Algorithm	Test 1	Test 2	Test 3	Overall result(us)
Stoer-Wagner's algorithm	392	375	420	396
Karger's algorithm	79718	79840	79519	79692(300 repeats)

Then we need to conclude the result of each algorithm to evaluate whether its performance reach our expectation

Stoer-Wagner's algorithm			
Test number	Time(us)	Vertex	Edges
1	493	4	4
2	455	8	12
3	530	21	32
4	467	10	45
5	396	10	13

The performance of Stoer-Wagner's algorithm seems to be stable for our tests. No matter a complicated test case or simple test case. And this result seems not match its expected time complexity  $O(|V||E| + |V|^2 \log |V|)$ . First reason I think is that we are in shortage of large and complex test case, which can effective distinguish the performance of Stoer-Wagner comparing to simple test case. Because precision of our test method is limited. It may not reflect the performance of Stoer-Wagner's algorithm with simple test case. Another reason is the logic of implementation may not totally reflect the logic of algorithm. Redundant loops and computation may significantly affect the performance of algorithm.

Now we turn to analyze Karger's algorithm in a similar way.

Karger's algorithm					
Test number	Time(us)	Vertex	Edges	Repeat	Time per repeat(us)
1	3204	4	4	32	100

2	32402	8	12	192	169
3	1894620	21	32	1764	1074
4	128046	10	45	300	426
5	79692	10	13	300	265

Although the result doesn't correlate with the expected time complexity  $O(|V|^2)$  from our results because of the logic structure of our implementation. But we can still find that time complexity of Karger's algorithm is related on number of vertices and edges. We can see in the test 3, the graph has 21 vertices and 32 edges. Karger's algorithm has the poorest performance. The running time is increasing with increase of complexity of the graph.

## Conclusion

From the result analysis in the previous section, we find that both implementation of algorithm can solve the minimum cut problem. Stoer-Wagner's algorithm give us a remarkable performance, which have stable running time for our all test cases. While the performance of Karger's algorithm rely on the number of vertices and edges in the graph. Besides, although single Karger's algorithm also has good performance, it need to repeat  $T = \frac{2}{n(n-1)} \ln n$  times to get reliable result by reducing the probability of getting wrong result into  $\frac{1}{n}$ , it doesn't have convincing performance in my experiment.

It should have some method to make both algorithm faster, which I haven't experiment all in my implementation. For Karger's algorithm since I apply adjacent list as data structure, its expected running time is  $O(|V|^2)$ . It should be ran faster in theory if I implement the algorithm like Kruskal's algorithm. Besides, another excellent improvement is using Karger-Stein algorithm. The essential idea is using several repeat of contraction until reach a state that the graph only has  $t$  vertices, and then using our original Karger's algorithm to handle rest of contraction. Remember we need to use  $T = \frac{2}{n(n-1)} \ln n$  times repeat in the original Karger's algorithm to get a reliable result, while in Karger-Stein result, normally we need repeat twice until  $t = 1 + \frac{n}{\sqrt{2}}$  vertices left in the graph, and pick one state that has minimum value

of minimum cut as our initial state. At this time we have over  $\frac{3}{4}$  probability that all edges in the minimum cut  $C$  are reserved at this state. And then apply original Karger's algorithm, repeat  $t=1+n/\sqrt{2}$  to do contraction upon the initial state, we can also get a reliable result of minimum cut. Although the probability to getting a correct minimum cut is slightly worse comparing to the result from original Karger's algorithm, the efficiency of Karger–Stein algorithm increase dramatically due to reduction of repeat.

For the Stoer-Wagner's algorithm, there is not much more space to increase its performance. However, we still can find advanced method about finding the maximum weight to determine most tightly connected vertex and updating the weight of edges. In the original implementation, we should determine weights of all the edges connected to current merged vertex set  $A$ , which needs  $O(|V|)$  time. And after contraction, we need to update the edges that connected to  $A$ , which needs  $O(|E|)$  time. We can build a Fibonacci heap here to handle these operation, it only needs  $O(\log |V|)$  amortized time and  $O(1)$  amortized time to finish weight searching and edge updating respectively.

## Reference

- [1] *Stoer–Wagner algorithm*. Wikipedia.org
- [2] *Karger's algorithm*. Wikipedia.org
- [3] Stoer M, Wagner F. *A simple min-cut algorithm* [J]. Journal of the ACM (JACM), 1997, 44(4): 585-591.
- [4] Karger D R, Stein C. *A new approach to the minimum cut problem* [J]. Journal of the ACM (JACM), 1996, 43(4): 601-640.
- [5] S. R. Arikati, K. Mehlhorn. *A correctness certificate for the Stoer-Wagner min-cut algorithm*. Information processing letters, 70, 1999, 251,-254.
- [6] Sanjoy Dasgupta CSE 103: *Probability and statistics Lecture note: Topic 4 — Randomized algorithms, II*. UC San Diego
- [7] Mehlhorn K, Uhrig C. *The minimum cut algorithm of Stoer and Wagner* [J]. 1995.