Algortithms for the Min-Cut Problem*

Illinois Institute of Technology
Applied Mathematics

Hongwei Jin
A20288745

May 14, 2013

*Talk topic in MATH 554, on Tuesday, March 30, 2013.

1 Introduction

1.1 Problem Definition

Let $G = (V, E)$ be undirected graph with n vertices, and m edges. We are interested in the notion of a cut in a graph.

Definition 1.1. A **cut** in G is a partiontion of the vertices of V into two sets S and T, $T = V(G)\backslash S$, where the edges of the cut are

$$(S, T) = \{uv | u \in S, v \in T, S \cap T = V(G), uv \in E(G)\}$$

where $S \neq \emptyset$ and $T \neq \emptyset$. We will refer to the number of edges in the cut (S, T) as the *size* of the cut.

We are intersted in the problem of computing the *minimum cut*, that is, the cut in the graph with minimum cardinality. Compute the cut with minimum number of edges in the graph. Namely, find $S \subseteq V$ such that (S, T) is as small as possible, and S is neither empty nor is T.

The problem actually comes in two flavors: in the *s-t min-cut problem*, we require that the two specific vertices s and t be on opposite sides of the cut; in what will be called the *min-cut problem*, or for emphasis the *global min-cut problem*, there is no specific two vertices. In this report, we will mainly discuss about global min-cut problem.

For an example of a minimum cut, see Figure 1.

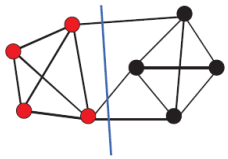


Figure 1: Min-cut example

The minimum cut problem has many applications, some of which are surveyed by Picard and Queyranne [1]. There are also many other applications:

- The problem of determining the connectivity of a network arises frequently in issues of network design and network reliability
- In information retrieval, minimum cuts have been used to identify clusters of topically related documents in hypertext systems
- Minimum cut problems also play an important role in large-scale combinatorial optimization
- Minimum cut problems arise in the design of compilers for parallel languages...

1.2 Previous Work

Several different approaches to finding minimum cuts have been investigated. Until recently, the most efficient algorithms used maximum flow computations. As the fastest known algorithms for maximum flow take $\Omega(mn)$ time, the best minimum cut algorithms inherited this bound. Recently, new and slightly faster approaches to computing minimum cuts without computing maximum flows have appeared. Parallel algorithms for the problem have also been investigated, but until now processor bounds have been quite large for unweighted graphs, and no good algorithms for weighted graphs were known.

The oldest known way to compute min-cut is to use their well known duality with max-flow [2]. Now we should recall some definition and theorem from graph theorem.

Definition 1.2. A **network** is a digraph with a nonnegative **capacity** $c(e)$ on each edge e and a distinguished source vertex s and sink vertex t. A **flow** f assigns a value $f(e)$ to each edge e. We write $f^+(v)$ for total flow on edges leaving v and $f^-(v)$ for total flow on edge entering v. A flow is **feasible** if it satisfies the **capacity constrains** $0 \leq f(e) \leq c(e)$ for each edge and the conservation constrains $f^+(v) = f^-(v)$ for each node $v \notin \{s, t\}$.
The value of a flow is the net flow into the sink vertex. A **maximum flow** is a feasible flow of maximum value.

Theorem 1.3 (Max-flow Min-cut Theorem (Ford and Fulkerson,1956)). *In every network, the maximum value of a feasible flow eqauls the minimum capacity of a source/sink cut.*

For an example, see Figure 2

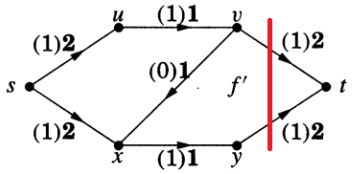


Figure 2: Max-flow Min-cut example

For combinatorial applications, Menger's Theorem can prove Max-flow Min-cut Theorem. Accutally, those two theorems can be prove from one to the other independently.

Computation of an s-t max-flow allows the immediate determination of an s-t min-cut. The best presently known sequential time bound for max-flow is $O(mn \log(n^2/m))$, found by Goldberg and Tarjan [3]. Global min-cut can be computed by minimizing over s-t max-flow; Hao and Orlin [4] show how the max-flow computations can be popelined so that together they take no more time than a single max-flow computation.

Recently, two approaches to finding minimum cuts without computing any maximum flows have appeared. One approach, developed by Gabow [5], is based on a matroid characterization of the minimum cut problem.

The second new approach uses no flow-based techniques at all. The central idea is to repeatedly identify and contractedges that are not in the minimum cut until the minimum cut becomes apparent. It applies only to undirected graphs, but they may be weighted. Nagamochi and Ibaraki [6] give a procedure called *scan-first search* that identifies and contracts an edge that is not in the minimum cut in $O(m + n \log n)$ time, which totally cost $O(mn + n^2 \log n)$.

2 Karger's Algorithm

Karger's algorithm is a randomized algorithm to compute a minimum cut of a connected undirected graph. It wes invented by David Karger when he was a PhD student at Stanford University, and first published in 1993 [7].

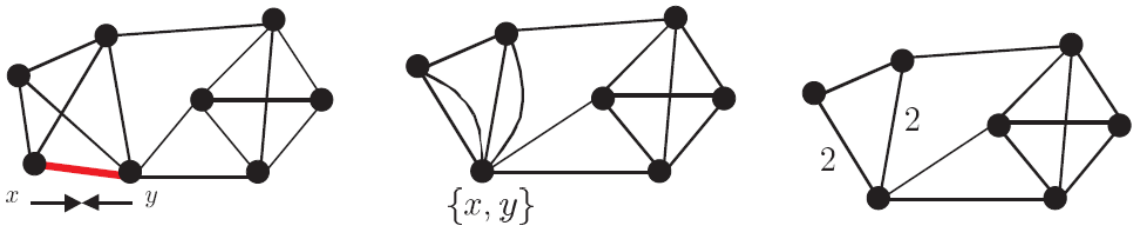
The idea of the algorithm is based on the concept of *contraction* of an edge uv in an undirected graph. Karger's basic algorithm iteratively contracts randomly chosen edges until only two nodes remain; those nodes represent a cut in the original graph. By iterating this basic algorithm a sufficient number of times, a minimum cut can be found with high probability. Thus it can be viewed as a Monte Carlo algorithm.

2.1 Contraction Algorithm

The fundamental idea of Karger's algorithm is a form of edge contraction.

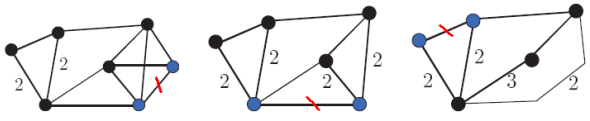
Definition 2.1. In a graph G, **contraction** of edge e with endpoints u, v is the replacement of u and v with single vertex whose incident edges are the edges other than e that were incident to u or v. the resulting graph,denoted as G/e, has one less edge that G.

Take an example in Figure 3

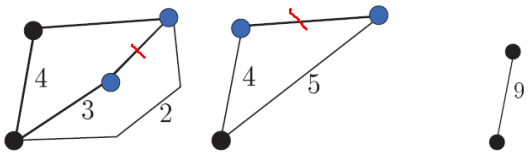


(a) Contract edge xy (b) merge into one vertex (c) after contraction

Figure 3: Contraction example



(a) contineous contraction



(b) end up with 2 vertex, but not min-cut

Figure 4: End with 2 vertex, it is a cut but NOT min-cut

The pseudocode are presented as below:

```
procedure MinCut (G)
while |V|>2
        choose e in E(G) unformly at random
        G = G/e
return the only cut in G
```

The edge contraction operation can be implemented in $T = n-2$ time for a graph with n vertices into 2 vertices. This is done by merging the adjacency lists of the two vertices being contracted, and then using hashing to do the fix-ups (i.e., we need to fix the adjacency list of the vertices that are connected to the two vertices). Note, that the cut is now computed counting multiplicities (i.e., if an edge is in the cut, and it has weight w, we add w to the weight of the cut.

Obvervation 2.2. The size of the minimum cut in G/e is at least as large as the minimum cut in G (as long as G/e has at least one edge). Since any cut in G/e has a corresponding cut of the same cardinality in G.

So, the main idea of our algorithm is to repeatedly perform contraction, which is beneficial since it shrinks the graph. And we would like to compute the cut in the resulting (smaller) graph. An extreme example of this, is shown in Figure 1.3, where we contract the graph into a single edge, which in turn corresponds to a cut in the original graph. (It might help the reader to think about each vertex in the contracted graph, as corresponding to a connected component in the original graph.)

Obvervation 2.3. Let $e_1, ...e_{n-2}$ be a sequence of edges in G, such that none of them is in the minimum cut, and such that $G' = G/e_1, ...e_{n-2}$ is a single multi-edge. Then, this multi-edge correspond to the minimum cut in G

Obvervation 2.4. The algorithm always output a cut, and the cut is not smaller than the minimum cut.

Obvervation 2.5. The algorithm runs in $O(n^2)$ time.

2.2 Algorithm Analysis

Lemma 2.6. A cut (S,T) is output by the Contraction Algorithm if and only if no edge crossing (S,T) is contracted by the algorithm.

Proof. The only if direction is obvious. For the other direction, consider two vertices on opposite sides of the cut (S,T). If they end up in the same meta-vertex, then there must be a path between them consisting of edges that were contracted. However, any path between them crosses (S,T), so an edge crossing cut (S,T) would have had to be contracted. This contradicts our hypothesis. □

Lemma 2.7. If a graph G has a minimum cut of size k, and it has n vertices, then $|E(G)| \geq \frac{kn}{2}$

Proof. Assume that each vertex degree is at least k, otherwise the vertex itself would form a minimum cut of size smaller than k. As such, there are at least $|E(G)| = \sum_{v \in V(G)} \deg(v)/2 \geq kn/2$ edges in the graph. □

Lemma 2.8. If we pick in random an edge e from a graph G, then with probability at most 2/n it belong to the minimum cut.

Proof. There are at least $kn/2$ edges in the graph and exactly k edges in the minimum cut. Thus the probability of picking an edge from the minimum cut is smaller then $\frac{k}{kn/2} = \frac{2}{n}$. □

Lemma 2.9. MinCut algorithm outputs the min cut in probability $\mathbb{P} \geq \frac{2}{n(n-1)}$

Proof. Let x_i be the event that edge e_i is not in the minimum cut of G_i. If the MinCut algorithm output a minimum cut, then all the event sequence $\{x_0, ...x_{n-3}\}$ will happen. Since at most with probability $2/n$ the edge will belong to the minimum cut. Thus we have the probability

$$(1 - \frac{2}{n})(1 - \frac{2}{n-1})...(1 - \frac{2}{3}) = (\frac{n-2}{n})(\frac{n-3}{n-1})...(\frac{1}{3}) = \frac{2}{n(n-1)}$$

□

Lemma 2.10. The probability that repeat MinCut algorithm $T = \binom{n}{2} \log n$ times fails to return the minimum cut is $< \frac{1}{n}$

Proof. The probability of failure is at most

$$(1 - \frac{2}{n(n-1)})^{\binom{n}{2} \log n} \leq \exp(-\log n) = \frac{1}{n}$$

□

Theorem 2.11. *One can compute the minimum cut in $O(n^4)$ time with constant probability to get a correct result. In $O(n^4 \log n)$ time the minimum cut is returned with high probability.*

3 Karger-Stein Algorithm

An extension of Karger's algorithm due to David Karger and Clifford Stein achieved an order of magnitude improvement [8]. They also call it *Recursive Contraction Algorithm*. The basic idea is to perform the contraction procedure until the graph reaches $\lceil 1 + n/\sqrt{2} \rceil$ vertices. Well, the probability of success in the first l iterations is

$$\mathbb{P}[x_0 \cap ... \cap x_{l-1}] \geq \prod_{i=0}^{l-1}(1 - \frac{2}{n-i}) = \prod_{i=0}^{l-1} \frac{n-i-2}{n-i} = \frac{n-2}{n}\frac{n-3}{n-1}... = \frac{(n-l)(n-l-1)}{n(n-1)}$$

Namely, this probability deteriorates very quickly toward the end of the execution, when the graph become small enough.

3.1 Recursive Contraction Algorithm

Obvervation 3.1. As the graph get smaller, the probability to make a bad choice increases. So, run the algorithm more times when the graph is smaller.

Thus according to the observation above, here present the pseudocode.

```
procedure MinCut(G,t)
while |V| > t
        choose e in E(G) unformly at random
        G = G/e
return the only cut in G
```

Lemma 3.2. The probability that Contract$(G, n/\sqrt{2})$ had NOT contracted the minimum cut is at least $1/2$.

Proof. Let $l = n - t = n - \lceil 1 + n/\sqrt{2} \rceil$, we will get

$$\mathbb{P}[x_0 \cap ... \cap x_{n-t}] \geq \frac{t(t-1)}{n(n-1)} = \frac{(\lceil 1 + n/\sqrt{2} \rceil)(\lceil 1 + n/\sqrt{2} \rceil - 1)}{n(n-1)} \geq \frac{1}{2}$$

□

Karger and Stein provide a new algorithm with recursive process, here is the provided pseudocode

```
procedure FastMinCut (G)
if |V| < 6
        t = 1+|V|/sqrt(2)
else
        G1 = MinCut(G,t)
        G2 = MinCut(G,t)
return min (FastMinCut(G1), FastMinCut(G2))
```

3.2 Algorithm Analysis

Lemma 3.3. The running time of FastMinCut(G) is $O(n^2 \log n)$, where $n = |V(G)|$.

Proof. Well, we perform two calls to Contract(G,t) which takes $O(n^2)$ time. And then we perform two recursive calls, on the resulting graphs. We have:

$$T(n) = O(n^2) + 2T(\frac{n}{\sqrt{2}})$$

The solution to this recurrence is $O(n^2 \log n)$ as one can easily verify. □

Theorem 3.4. *Running FastMinCut finds the minimum cut with probability larger that* $2 \log 2 / \log n$*,which can be notated as* $\Omega(1/\log n)$

Proof. Let P(n) be the probability that the algorithm successes on a graph with n vertices.

The probability to succeed in the first call on G_1 is the probability that contract did not hit the minimum cut (this probability is larger than 1/2), times the probability that the algorithm succeeded on G_1 in the recursive call (those two events are independent). Thus, the probability to succeed on the call on G_1 is at least $\frac{1}{2}P(\frac{n}{\sqrt{2}})$. Thus, the probability to fail on G_1 is $\leq 1 - \frac{1}{2}P(\frac{n}{\sqrt{2}})$.

The probability to fail on both G_1 and G_2 is smaller than

$$(1 - \frac{1}{2}P(\frac{n}{\sqrt{(2)}}))^2$$

And thus, the probability for the algorithm to succeed is

$$P(n) \geq 1 - (1 - \frac{1}{2}P(\frac{n}{\sqrt{(2)}}))^2 = P(\frac{n}{\sqrt{(2)}}) - \frac{1}{4}(P(\frac{n}{\sqrt{(2)}}))^2$$

We need to solve this recurrence. Divide both sides of equation by $P(n/\sqrt{2})$ we have

$$\frac{P(n)}{P(n/\sqrt{2})} \geq 1 - \frac{1}{4}P(\frac{n}{\sqrt{2}})$$

It is now easy to verify that this inequality holds for $P(n) \geq c/\log n$, since the worst case is $P(n) = c/\log n$ we verify this inequality for this value. Indeed,

$$\frac{c/\log n}{c/\log(n/\sqrt{2})} \geq 1 - \frac{c}{4\log(n/\sqrt{2})}$$

$$\Rightarrow \frac{\log n - \log\sqrt{2}}{\log n} \geq \frac{4(\log n - \log\sqrt{2}) - c}{4(\log n - \log\sqrt{2})}$$

Let $\Delta = \log n$

$$\Rightarrow \frac{\Delta - \log\sqrt{2}}{\Delta} \geq \frac{4(\Delta - \log\sqrt{2}) - c}{4(\Delta - \log\sqrt{2})}$$

$$\Rightarrow 4(\Delta - \log\sqrt{2})^2 \geq 4\Delta(\Delta - \log\sqrt{2}) - c\Delta$$

$$\Rightarrow c\Delta - 4\Delta\log\sqrt{2} + 4\log^2\sqrt{2} \geq 0$$

Which clear hold for $c \geq 4 \log \sqrt{2}$

We conclude, that the algorithm succeeds in finding the minimum cut in probability $\geq 2\frac{\log 2}{\log n}$, which can be notated as $\Omega(1/\log n)$. □

According to the theorem above, running the FastMinCut $c \log^2 n$ times, guarrantee that the algorithm outputs the minimum cut with probability $\geq 1 - \frac{1}{n^2}$, c is a constant large enough.

Theorem 3.5. *With high probability we can find all min cuts in the running time of* $O(n^2 \log^3 n)$.

Proof. Since We know that $P(n) = O(\frac{1}{\log n})$, therefore after running this alforithm $O(\log^2 n)$ times, the probability of missing a specific min-cut is

$$\mathbb{P}[\text{miss a specific min} - \text{cut}] = (1 - P(n))^{O(\log^2 n)} \leq (1 - \frac{c}{\log n})^{3 \log^2 n / c} \leq \exp(-3 \log n) = \frac{1}{n^3}$$

And there are at most $\binom{n}{2}$ min-cuts, hence the probability of missing any min-cut is

$$\mathbb{P}[\text{miss any min} - \text{cut}] \leq \binom{n}{2} \frac{1}{n^3} = O(\frac{1}{n})$$

. Thus the probability of success is large as n is large enough. □

4 Implementation

To specific the Min-Cut problem, here we provide an example with 200 vertices. It provides codes with Python, such that we can understand the algorithms thoroughly. Figure 5 shows the original graph, all the vertices lies on the circle.

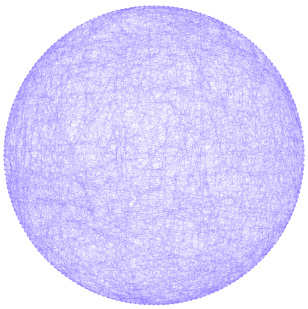


Figure 5: 200-vertices simple graph

The example has the following properties:

```
Total edges: 2517
Total vertices: 200
Maximum degree: 39
Minimum degree: 20
average degree: 25
Mincut is 17
```

For the simple MinCut algorithm, the codes are implemented as follows:

```
def MinCut(graph, t):
    while len(graph) > t:
        start = random.choice(graph.keys())
        finish = random.choice(graph[start])

    # # Adding the edges from the absorbed node:
        for edge in graph[finish]:
            if edge != start:  # this stops us from making a self-loop
                graph[start].append(edge)

    # # Deleting the references to the absorbed node and changing them to the
                                                source node:
        for edge1 in graph[finish]:
            graph[edge1].remove(finish)
            if edge1 != start:  # this stops us from re-adding all the edges in
                                                        start.
                graph[edge1].append(start)
        del graph[finish]

    # # Calculating and recording the mincut
    mincut = len(graph[graph.keys()[0]])
    cuts.append(mincut)
    return graph
```

By the analysis in Chapter 2, run this algorithm $n^2 \log n$ times, thus we can get the min-cut with high probability.

```
filename = "KargerMinCut.txt"
file1 = open(filename)
graph = {}
cuts = []
edge_num = 0
edge_list = []
for line in file1:
    node = int(line.split()[0])
    edges = []
```

```
        for edge in line.split()[1:]:
            edges.append(int(edge))
        graph[node] = edges
        edge_num = edge_num + len(edges)
        edge_list.append(len(edges))
    file1.close()
    count = len(graph) * len(graph) * int(math.log(len(graph))) # running times
    while i++ < count:
        graph1 = copy.deepcopy(graph)
        g = MinCut(graph1,2)
```

Figure 6 shows the corelated result, the number of edges between S and T is 17, which indicate the minimum cut size.

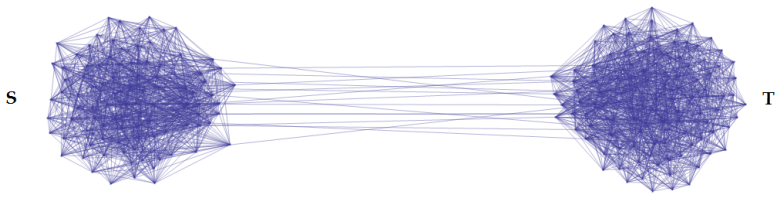


Figure 6: Partition into two partite with MinCut=17

For Karger Stein algorithm, we implemented as recursive way, codes show as follows:

```
def FastMinCut(graph):
    if len(graph) < 6:
        return MinCut(graph,2)
    else :
        t = 1 + int(len(graph) / math.sqrt(2))
        graph_1 = MinCut(graph, t)
        graph_2 = MinCut(graph, t)
        if len(graph_1) > len(graph_2):
            return FastMinCut(graph_2)
        else:
            return FastMinCut(graph_1)

count = int(math.log(len(graph))) * int(math.log(len(graph))) # running times
  while i++ < count:
      graph1 = copy.deepcopy(graph)
      g = FastMinCut(graph1)
```

It outputs the same result, while consumes much less time.

There is a paper analysis most recent algorithm to find min-cut by conducting experimental evaluation the relative performance of these algorithms rather than theoretical analysis [9].

5 Conclusion

First, let take a look of the comparision of some classical attempt to figure out the min-cut problem.

- *s-t min-cut problem*

Bound	**unweighted graph**	**weighted graph**
Directed	$cm \log \frac{n^2}{m}$ [5]	$mn \log \frac{n^2}{m}$ [4]
Undirected	$c^2 n \log \frac{n}{c}$ [5]	$mn + n^2 \log n$ [6]

- *global min-cut problem*

Bound	**unweighted graph**	**weighted graph**
Directed	$\mathcal{P} - complete$	$\mathcal{P} - complete$
Undirected	$O(n^2 \log^3 n)$ - Karger-Stein [8]	$O(n^2 \log^3 n)$ - Karger-Stein [8]

Then take a look at comparision of Karger's algorithm and Karger-Stein algorithm.

Bound	**Karger algorithm**	**Karger-Stein algorithm**
Probability	$O(1/n^2)$	$O(1/\log n)$
Cost	$O(n^2)$	$O(n^2 \log n)$
Running times	$\binom{n}{2} \log n$	$\log^2 n$
Totol Order	$O(n^4 \log n)$	$O(n^2 \log^3 n)$

From the table above, we can get that Karger-Stein Algorithm gets quite close to the minimum order $O(n^2)$. Since one has to touch every edge in the graph at least once in a dense graph.

While there still several ways to improve the algorithm. Main ideas of improvement can be classified as two respects: 1) reduce the running times, thus we can still get the minimum cut with high probability; 2) reduce the probability of contract the graph ending with a min-cut.

Parallel algorithms for the minimum cut problem have also been explored, though with much less satisfactory results. The parallel version of contraction algorithm runs in polylogarithmic time using n^2 processors on a PRAM. It thus provides the first proof that the minimum cut problem with arbitrary edge weights can be solved in $\mathcal{RNC}$. In a contrasting result, they showed that the directed minimum cut problem is $\mathcal{P}$-complete and thus appears unlikely to have an $\mathcal{RNC}$ solution. $\mathcal{RNC}$ is the class of problems that can be solved by a randomized algorithm in polylogarithmic time using a PRAM with a polynomial number of processors. In Karger's original paper [7], he also provide a parallel algorithm $\mathcal{RNC}$, such that we can find out the min-cut in $O(n^2)$ time. In the later paper he also proved that $\mathcal{RNC}$ works.

They provide a compact algorithm to implement the parallel algorithms. The idea is using the permutation of edges: instead of choosing edges one at a time, we begin by generating a random permutation L of the edges according to the uniform distribution. Imagine contracting edges in the order in which they appear in the permutation, until only two vertices remain. This is clearly

equivalent to the abstract formulation of the Contraction Algorithm. We can immediately deduce that with probability $\Omega(n^{-2})$, a random permutation will yield a contraction to two vertices that determine a particular minimum cut.

```
Procedure Compact(G,L,k)
input: A graph G, list of edges L, and parameter k
if G has k vertices or L is not emptyset
        return G
else
        Let L_1 and L_2 be the first and second halves of L
        Find the connected components in graph H = (V,L_1)
        if H has fewer than k components
                return Compact(G, L_1, k)
        else
                return Compact(G/L_1, L_2/L_1, k)
```

Recently, some scientist also provide some new random algorithm to find the minimum cut. Timo Ktzing et al. apply "Ant colony optimization method" can obtain the solution in expected polynomial time [10].Ant Colony Optimization (ACO) is a powerful metaheuristic for solving combinatorial optimization problems. However they also show that high use of pheromones has a negative effect, and the ACO algorithm may get trapped in local optima resulting in an exponential runtime to obtain an optimal solution. Frank Neumann et al. apply "Randomized Search Heuristics" method to obtain the solution in expected polynomial time [11]. They develop a bi-criteria approach based on the famous maximum-flow minimum-cut theorem that enables evolutionary algorithms to find an optimal solution in expected polynomial time.

Randomized algorithms are widely used in the optimization of combinatorics and graph theory. You can refer to a classical textbook *Randomized Algorithm*[1995, Rajeev Motwani,Prabhakar Raghavan [12]], the prime author is also the advisor of David Karger. An supplementary reading material is class notes organized by Sariel Har-Peled [13].

Thanks to Lujia Wang for helping me figure out some problems.

References

[1] Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Mathematical Programming*, 22(1):121–121, 1982.

[2] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[3] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[4] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.

[5] Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 112–122. ACM, 1991.

[6] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[7] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 21–30. Society for Industrial and Applied Mathematics, 1993.

[8] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[9] Chandra S Chekuri, Andrew V Goldberg, David R Karger, Matthew S Levine, and Cliff Stein. Experimental study of minimum cut algorithms. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 324–333. Society for Industrial and Applied Mathematics, 1997.

[10] Timo Kötzing, Per Kristian Lehre, Frank Neumann, and Pietro Simone Oliveto. Ant colony optimization and the minimum cut problem. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1393–1400. ACM, 2010.

[11] Frank Neumann, Joachim Reichel, and Martin Skutella. Computing minimum cuts by randomized search heuristics. *Algorithmica*, 59(3):323–342, 2011.

[12] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[13] Sariel Har-Peledx. Class notes for randomized algorithms, 2005.