Karger's Algorithm

Dhanush Krishna R AM.EN.U4AIE20021

Pranav Jayasankar Nair AM.EN.U4AIE20056

Problem Statement:

Given an undirected and unweighted graph, find the smallest cut (smallest number of edges that disconnects the graph into two components). The input graph may have parallel edges. This problem is called the minimum cut problem. The two main algorithms used to solve this problem are the Max-flow based min cut algorithm and Karger's algorithm for minimum cut.

Working:

A *cut* (S,T) in an undirected graph G=(V,E) is a partition of the vertices V into two non-empty, disjoint sets S U T = V. The main difference between the two algorithms is that the Max- flow algorithm provides a deterministic approach whereas Karger's algorithm provides a probabilistic approach to the same problem. So why use a probabilistic algorithm over a deterministic one?

The max- flow min cut algorithm comes at the cost of having an optimization of O(|V||E|^2) where V is the number of vertices and E is the number of sides whereas Karger's random algorithm has an optimization of O(V^2). Karger's algorithm uses random sampling so solve the min-cut problem. Karger's algorithm uses the principle of contraction on the edges of the graph. The process of contraction involves bringing together the 2 nodes on either side of an edge and merge them into one node. By repeating this process, we are left with a graph with only 2 vertices (effectively) . The cut between the 2 vertices performed on the expanded (initial) form of the graph is used to determine which the minimum cut would be. Probability that the cut produced by Karger's Algorithm is Min-Cut is greater than or equal to 1/(n^2) and has an optimization of O(n^2). However, this is not enough, so the probability is increased by performing multiple runs of the algorithm. If we run Karger's algorithm n^2 times, the probability of failure that we don't get a minimum cut is 1/e. A constant which no longer depends on the number of vertices.

However, if we run Karger's algorithm $n^2 \ln(n)$ times, which is regarded as the optimal value, then P(fail) <= 1/n, and P(success) >= 1–1/n, which means as you get bigger and bigger graph , the probability of success of Karger's algorithm gets better. So, in a graph with 50 nodes, if we run the algorithm around 10,000 times, the probability of success is 98%.

Such a good success rate of 98% comes with a cost though. The running time is polynomial in n (the number of vertices) and m (the number of edges), which is pretty slow: $O(n^2m)$. A slightly altered version, Karger-Stein algorithm, has twice the success probability as the original algorithm.

During the sequence of edge contractions, the probability of choosing right edges is high at the beginning but low later since there are small number of vertices left. The Karger-Stein algorithm uses the general framework of that in the Karger's mincut algorithm but runs the "later" part more times. Through this method, the order is reduced from the default O(n^4 log n) to O(n^2 log^3 n).

An extension of Karger's algorithm due to David Karger and Clifford Stein achieved an order of magnitude improvement They also call it Recursive Contraction Algorithm. The basic idea is to perform the contraction procedure until the graph reaches 1 + n/√2 vertices. The probability of success in the first l iterations deteriorates very quickly toward the end of the execution, when the graph become small enough. As the graph get smaller, the probability to make a bad choice increases. So, run the algorithm more times when the graph is smaller. Judging by the the pseudocode below, the probability that Contract(G, n/√2) had NOT contracted the minimum cut is at least ½. Taking this into consideration, Karger and Stein provide a new algorithm with recursive process.

Uses:

The min cut algorithm can be applied in processes like image segmentation, if you want to break an image into segments, easiest way is to break links with low affinity, i.e., minimum number of cuts. Think of A polygon mesh. A polygon mesh is the collection of vertices, edges, and faces that make up a 3D object. Breaking images links with low affinity can give you separate distinct images. Other applications include:

1) In war situation, a party would be interested in finding minimum number of links that break communication network of enemy.

2) The min-cut problem can be used to study reliability of a network (smallest number of edges that can fail).

Algorithm

1) Initialize contracted graph CG as copy of original graph
2) While there are more than 2 vertices.
    a) Pick a random edge (u, v) in the contracted graph.
    b) Merge (or contract) u and v into a single vertex (update
       the contracted graph).
    c) Remove self-loops
3) Return cut represented by two vertices.

The algorithm

Given a graph G = (V, E), for an edge e = (v, w) ∈ E, we call the following action as contracting e to obtain a new graph, denoted by G/e:

1. Replace vertices v and w by new vertex y.

2. For all edges (v, z) ∈ E in original graph G, replace by (y, z); for all edges (w, z) ∈ E, replace by (y, z).
3. Drop all edges (v, w) ∈ E.

Note that G/e is a multi-graph without self-loops. We have the following observations on edge contractions: For a given graph G = (V, E) and e = (v, w) ∈ E, let G' = G/e. Consider cuts in G and cuts in G' , then every cut in G' corresponds to a cut in G. For a subset S ⊂ V , where e ∈ S, then δG(S) = δG' (S). Now, consider contracting a sequence of edges e1, e2, . . . , en−2. Since the number of vertices decreases by one via each contraction, there are two big meta-vertices left. Note that the (multi-)edges between those two vertices represent the cut(S ' , S^{-}') in G, where the vertices in S' and S^{-}' are contracted into those two big meta-vertices respectively. Fix a cut (S * , S^{-}*) of minimum size |δ(S *)| = k. We want all of these edges e1, e2, . . . , en−2 ∈/ δ(S *). In a randomized algorithm, just take guesses at choosing edges. The minimum cut is small, so if you choose a random edge from the whole graph, it is unlikely that it will be in the cut. Below is the formal algorithm.

Algorithm 1: Karger's min-cut algorithm input :

input : Graph G = (V, E).

n ← |V (G)|, C ← E;

for i ← 1 to l(n/2) do

    G' ← G;

    Repeat

        Choose an edge e ∈ E(G');

        G' ← G'/e;

    until |V (G')| = 2;

    if |C| ≥ |E(G')| then

        C ← E(G')

output: C

The Karger-Stein algorithm

During the sequence of edge contractions, the probability of choosing right edges is high at the beginning but low later since there are small number of vertices left. The Karger-Stein algorithm uses the general framework of that in the Karger's mincut algorithm but runs the "later" part more times.

Suppose we contract edges from n vertices to l vertices twice. We can choose l to have the probability of success is about 1/2 . So in expectation, one of the two runs work. Then we recurse each of these bifurcates. Take best of two, then pop back up from the recursion.

Algorithm 2:

The Karger-Stein algorithm: KargerStein(G)

input : Graph G with n vertices

if $n \geq 6$ then

Do 2 runs and output the best of the 2:

Use Karger's mincut algorithm to contract edges until $\sqrt{n}/2 + 1$ vertices are left, denoted as G';

Recurse KargerStein(G').

Note that 6 is chosen in the algorithm as $\sqrt{6}/2 + 1 \geq 2$.

How to increase probability of success?
The above probability of success of basic algorithm is very less. For example, for a graph with 10 nodes, the probability of finding the min-cut is greater than or equal to 1/100. The probability can be increased by repeated runs of basic algorithm and return minimum of all cuts found.

Input and Output

Input : Undirected and Unweighted graph

Output: Cut found by Karger's algorithm

c is number of edges in min-cut

m is total number of edges

n is total number of vertices

S_1 = Event that one of the edges in $\{e_1, e_2, e_3, .. e_c\}$ is chosen in 1st iteration.

S_2 = Event that one of the edges in $\{e_1, e_2, e_3, .. e_c\}$ is chosen in 2nd iteration.

S_3 = Event that one of the edges in $\{e_1, e_2, e_3, .. e_c\}$ is chosen in 3rd iteration.

.....

The cut produced by Karger's algorithm would be a min-cut if none of the above

events happen.

So the required probability is $P[S_1' \cap S_2' \cap S_3' \cap .....]$

Probability that a min-cut edge is chosen in first iteration:

Let us calculate $P[S_1']$

$P[S_1] = c/m$

$P[S_1'] = (1 - c/m)$

Above value is in terms of m (or edges), let us convert it in terms of n (or vertices) using below 2 facts..

1) Since size of min-cut is c, degree of all vertices must be greater

than or equal to c.

2) As per Handshaking Lemma, sum of degrees of all vertices = 2m

From above two facts, we can conclude below.

$n*c <= 2m$

$m >= nc/2$

$P[S_1] <= c / (cn/2)$

$<= 2/n$

$P[S_1] <= c / (cn/2)$

$<= 2/n$

$P[S_1'] >= (1-2/n)$ ----------(1)

Probability that a min-cut edge is chosen in second iteration:

$P[S_1' \cap S_2'] = P[S_2' \mid S_1'] * P[S_1']$

In the above expression, we know value of $P[S_1'] >= (1-2/n)$

$P[S_2' \mid S_1']$ is conditional probability that is, a min cut is

not chosen in second iteration given that it is not chosen in first iteration

Since there are total (n-1) edges left now and number of cut edges is still c,

we can replace n by n-1 in inequality (1). So we get.

$P[S_2' \mid S_1'] >= (1 - 2/(n-1))$

$P[S_1' \cap S_2'] >= (1-2/n) \times (1-2/(n-1))$

Probability that a min-cut edge is chosen in all iterations:

$P[S_1' \cap S_2' \cap S_3' \cap ..... \cap S_{n-2}']$

$>= [1 - 2/n] * [1 - 2/(n-1)] * [1 - 2/(n-2)] * [1 - 2/(n-3)] *...$

$... * [1 - 2/(n - (n-4)] * [1 - 2/(n - (n-3)]$

$>= [(n-2)/n] * [(n-3)/(n-1)] * [(n-4)/(n-2)] * .... 2/4 * 2/3$

$>= 2/(n * (n-1))$

$>= 1/n^2$

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
```

```
procedure fastmincut(G = (V, E)):
if |V| ≤ 6:
    return mincut(V)
else:
    t ← ⌈1 + |V|/√2⌉
    G1 ← contract(G, t)
    G2 ← contract(G, t)
    return min {fastmincut(G1), fastmincut(G2)}
```

Comparison between Karger 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	$O(n^4 \log n)$	$O(n^2 \log^3 n)$