Minumum Cut

Karger and Stein's randomized Algorithm

Advanced Algorithms

Davide Bresolin

2021/2022

Minimum cut: Karger's algorithm

```
function Full_Contraction(G = (V, E))
    for i ← 1 to |V| − 2 do
        e ← Random(E)
        G′ = (V′, E′) ← G/e
        V ← V′
        E ← E′
    return |E|
```

```
function Karger(G, k)
    min ← +∞
    for i ← 1 to k do
        t ← Full_Contraction(G)
        if t < min then
            min ← t
    return min
```

- Full_Contraction finds a minimum cut with probability $2/n^2$
- if you run Full_Contraction $n^2 \log n$ times, error probability is smaller than $1/n$
- total running time is $O(n^4 \log n)$

Minimum cut for weighted graphs

Problem

Given a weighted undirected graph $G = (V, E, w)$ with positive weights, and a cut (S, T) of G, the weight of the cut (S, T) is defined as

$$w(S, T) = \sum_{e \in E(S,T)} w(e).$$

The global min-cut problem is the problem of finding a cut of G with the smallest weight.

- Karger's algorithm is defined for unweighted graphs
- How can we extend it to weighted graphs?
- How can we make it more efficient?

Full_Contraction for weighted graphs

function Full_Contraction($G = (V, E, w)$)
  for $i \leftarrow 1$ to $|V| - 2$ **do**
    choose an edge $e \in E$ with probability proportional to $w(e)$
    $G' = (V', E') \leftarrow G/e$
    $V \leftarrow V'$
    $E \leftarrow E'$
  return $\sum_{e \in E} w(e)$

The properties of Full_Contraction do not change:

- Full_Contraction finds a minimum cut with probability $2/n^2$
- if you run Full_Contraction $n^2 \log n$ times, error probability is smaller than $1/n$
- total running time is $O(n^4 \log n)$

How to implement Full_Contraction

We can implement Full_Contraction with $O(n^2)$ running time

- Data structures:
  - W: weighted adjacency matrix $n \times n$ such that

$$W[u, v] = \begin{cases} w(u, v) & \text{if } (u, v) \in E \\ 0 & \text{if } (u, v) \notin E \end{cases}$$

  - D: weighted degree of the vertices:

$$D[u] = \sum_{v \in V} W[u, v]$$

- Two steps:
  1. Pick an edge
  2. Contraction

How to pick an edge (1)

Given m edges $e_1, \ldots, e_m$ with weigths $w_1, \ldots, w_m$:

- build the cumulative weights vector:

$$C[k] = \sum_{i=1}^{k} w_i$$

- choose an integer r uniformly at random from $0, \ldots, C[m]$
- use binary search to find the edge e_i such that $C[i-1] \leq r < C[i]$

We call the above procedure Random_Select(C)

- C can be any cumulative weight array (non necessarily of edges)

How to pick an edge (2)

Edge_Select(D, W)

1. Choose u with probability proportional to $D[u]$
   - build the cumulative weights array of $D[u]$:

     $$C[k] = \sum_{i=1}^{k} D[i]$$

   - call Random_Select to select the first endpoint u
2. Once u is fixed, choose v with probability proportional to $W[u, v]$
   - build the cumulative weights array of $W[u, v]$:

     $$C[k] = \sum_{i=1}^{k} W[u, i]$$

   - call Random_Select to select the second endpoint v
3. return the edge (u, v)

Contracting and edge

function Contract_Edge(u,v)

$D[u] \leftarrow D[u] + D[v] - 2W[u,v]$

$D[v] \leftarrow 0$

$W[u,v] \leftarrow W[v,u] \leftarrow 0$

for each vertex $w \in V$, except u and v **do**

$W[u,w] \leftarrow W[u,w] + W[v,w]$

$W[w,u] \leftarrow W[w,u] + W[w,v]$

$W[v,w] \leftarrow W[w,v] \leftarrow 0$

Recursive contraction algorithm

function Contract($G = (D, W), k$)
  $n \leftarrow$ number of vertices in G
  for $i \leftarrow 1$ to $n - k$ **do**
    $(u, v) \leftarrow$ Edge_Select(D, W)
    Contract_Edge(u, v)
  return D, W

- This function returns a contraction of G to k vertices
- The graph G is represented by the matrix W and the vector D

Karger and Stein's contraction algorithm

function Recursive_Contract($G = (D, W)$)
    $n \leftarrow$ number of vertices in G
    if $n \leq 6$ **then**
        $G' \leftarrow$ Contract($G, 2$)
        return weight of the only edge (u, v) in G'
    $t \leftarrow \lceil n/\sqrt{2} + 1 \rceil$
    for $i \leftarrow 1$ to 2 **do**
        $G_i \leftarrow$ Contract(G, t)
        $w_i \leftarrow$ Recursive_Contract(G_i)
    return $\min(w_1, w_2)$

Recursive_Contract: properties

- Recursive_Contract has $O(n^2 \log n)$ running time
- Recursive_Contract finds a minimum cut with probability $1/\log n$
- by repeating Recursive_Contract $\log^2 n$ times, the error probability became less or equal to $1/n$
- the total running time of the algorithm is $O(n^2 \log^3 n)$