Advanced Algorithm

Jialin Zhang
zhangjialin@ict.ac.cn

Institute of Computing Technology, Chinese Academy of Sciences

March 21, 2019

A Randomized Algorithm for Min-Cut Problem

Contrast Algorithm:

1. Pick an edge uniformly at random;
2. Merge the endpoints of this edge;
3. Remove self-loops;
4. Repeat steps 1-3 until there are only two vertices remain.
5. The remaining edges form a candidate cut.

Min Cut

- Successful probability: $\Omega(\frac{1}{n^2})$.
- Time complexity: $O(n^2)$.
- Improvement?
  - FastCut algorithm
  - Ref: Randomized Algorithm - Chapter 10.2.
  - Algorithm FastCut runs in $O(n^2 \log n)$ time and uses $O(n^2)$ space.
  - Successful Probability is $\Omega(\frac{1}{\log n})$.

Homework

1. Prove the successful probability for FastCut algorithm is $\Omega(\frac{1}{\log n})$.
2. Randomized Algorithm - Exercise 10.9, Page 293.

Homework (optional)

We define a k-way cut-set in an undirected graph as a set of edges whose removal breaks the graph into k or more connected components. Show that the randomized min-cut algorithm can be modified to find a minimum k-way cut-set in $O(n^{2(k-1)})$ time.
Hints:

1. When the graph has become small enough, say less than $2k$ vertices, you can apply a deterministic k-way min-cut algorithm to find the minimum k-way cut-set without any cost.
2. To lower bound the number of edges in the graph, one possible way is to sum over all possible trivial k-way, i.e. $k-1$ singletons and the complement, and count how many times an edge is (over)-counted.

Lecture 2.1: Complexity Class

Las Vegas VS. Monte Carlo

- Ref: Randomized Algorithm - Chapter 1.2
- Las Vegas algorithm
  - ex. Quick Sort
  - random running time
- Monte Carlo algorithm
  - randomized Min-cut algorithm
  - random quality of solution
  - for decision problem: one-side error, two-side error

Complexity Classes

- Ref: Randomized Algorithm - Chapter 1.5
- We only consider decision problem in this class.

Definition (Language)

A language $L \subseteq \Sigma^*$ is any collection of strings over Σ.

Usually $\Sigma = \{0, 1\}$, and Σ^* is the set of all possible strings over this alphabet.

Definition (P)

$L \in P \Leftrightarrow \exists$ polynomial time algorithm A s.t.

$$\forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow A(x) \text{ accepts} \\ x \notin L \Rightarrow A(x) \text{ rejects} \end{cases}$$

Complexity Class: NP

Definition (NP)

$L \in NP \Leftrightarrow \exists$ polynomial time algorithm A s.t.

$$\forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow \exists y, |y| = \mathrm{poly}(|x|), A(x,y) \text{ accepts} \\ x \notin L \Rightarrow \forall y, |y| = \mathrm{poly}(|x|), A(x,y) \text{ rejects} \end{cases}$$

Definition (co-NP)

co-NP $= \{\bar{L} \mid L \in NP\}$

- Open problem: NP $=$ P?
- more classical complexity class: EXP, PSPASE, L, #P, etc ...

Famous NP-complete problems

- NP-hard problem(informal definition): A is NP-hard $\Leftrightarrow$ if A is polynomial time solvable, all problems in NP are polynomial time solvable.
- NP-complete problem: if A is NP-hard and $A \in$NP.
- Famous NP-complete problems
    - 3-SAT
    - Vertex cover, Set cover
    - Clique, Independent set
    - Hamilton cycle, Traveling salesman problem
    - Integer programming

Randomized Complexity Class: RP and co-RP

Definition (RP(Randomized Polynomial time))

$L \in RP \Leftrightarrow \exists$ randomized algorithm A running in worst-case polynomial time, s.t.

$$\forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow Pr(A(x) \text{ accepts}) \geq 1/2 \\ x \notin L \Rightarrow Pr(A(x) \text{ accepts}) = 0 \end{cases}$$

Definition (co-RP)

$L \in$ co-RP $\Leftrightarrow \exists$ randomized algorithm A running in worst-case polynomial time, s.t.

$$\forall x \in \Sigma^* \begin{cases} x \in L \Rightarrow Pr(A(x) \text{ accepts}) = 1 \\ x \notin L \Rightarrow Pr(A(x) \text{ accepts}) \leq 1/2 \end{cases}$$

Theorem

(1) $RP \subseteq NP$; (2) $co\text{-}RP \subseteq co\text{-}NP$;

Randomized Complexity Class: BPP, PP

Definition (BPP(Bounded-error Probabilistic Polynomial time))

$L \in BPP \Leftrightarrow \exists$ randomized algorithm A running in worst-case polynomial time, s.t.

$$\forall x \in \Sigma^* \begin{cases} x \in L \Rightarrow Pr(A(x) \text{ accepts}) \geq 3/4 \\ x \notin L \Rightarrow Pr(A(x) \text{ accepts}) \leq 1/4 \end{cases}$$

Definition (PP(Probabilistic Polynomial time))

$L \in PP \Leftrightarrow \exists$ randomized algorithm A running in worst-case polynomial time, s.t.

$$\forall x \in \Sigma^* \begin{cases} x \in L \Rightarrow Pr(A(x) \text{ accepts}) > 1/2 \\ x \notin L \Rightarrow Pr(A(x) \text{ accepts}) < 1/2 \end{cases}$$

Theorem

(1) $RP \subseteq BPP \subseteq PP$; (2) $NP \subseteq PP$.

Randomized Complexity Class: ZPP

Definition (ZPP: Zero-error Probabilistic Polynomial time)

The class ZPP is the class of languages that have Las Vegas algorithms running in expected polynomial time.

Theorem

ZPP $=$ *RP* $\cap$ *co-RP*.

Open problems

1. $\mathrm{RP} \overset{?}{=} \text{co-RP}$
2. $\mathrm{RP} \overset{?}{\subseteq} \mathrm{NP} \cap \text{co-NP}$
3. $\mathrm{BPP} \overset{?}{\subseteq} \mathrm{NP}$
4. $\mathrm{BPP} = \mathrm{P}$?

Relations between different complexity classes:

$$P \subseteq RP \begin{array}{l} \subseteq NP \\ \subseteq BPP \end{array} \subseteq PP \subseteq PSPACE$$

Homework

1. Randomized Algorithm - Exercise 1.10, Page 22.
2. Randomized Algorithm - Problem 1.13, Page 27.
3. (Optional) Randomized Algorithm - Problem 1.15, Page 27.

Lecture 2.2: More examples of randomized algorithms

Matrix Multiplication Verification

- Given three matrixes $A, B, C \in \{0,1\}^{n\times n}$, verify $A \times B = C$.
- Deterministic matrix multiplication
  - $O(n^{2.3728639})$, by Francois Le Gall, 2014
  - $O(n^{2.3729})$, by Virginia Vassilevska Williams, 2013
  - $O(n^{2.376})$, by Don Coppersmith and Shmuel Winograd, 1990
- Randomized algorithm for verification
  - Algo: Randomly choose vector v, test $A \cdot B \cdot v = C \cdot v$?
  - co-RP
  - Monte Carlo Algorithm. Successful probability?

EQ problem in Communication Complexity

- Let $x, y \in \{0,1\}^n$, define $EQ(x,y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$
- EQ problem: Alice holds x, Bob holds y. If they want to decide $EQ(x,y)$, how many bits do they need to communicate with each other?
- Deterministic communication complexity: $\Omega(n)$.
- Randomized algorithm
  - Algorithm:
    1. Define $f(z) = x_0 + x_1 z + \cdots + x_{n-1} z^{n-1}$, $g(z) = y_0 + y_1 z + \cdots + y_{n-1} z^{n-1}$ over F_p. p is a large prime;
    2. Alice randomly chooses $z \in \{0, 1, \cdots, p-1\}$, then send $z, f(z)$ to Bob;
    3. Bob tests if $f(z) = g(z)$.
  - co-RP
  - Monte Carlo Algorithm. Successful probability?