Advanced Core in Algorithm Design #13
算法設計要論 第 13 回

Yasushi Kawase
河瀬 康志

Jan. 10th, 2023

last update: 12:17pm, January 10, 2023

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

1. Randomized Quick Sort
2. Minimum Cut Problem
3. Identity Testing
4. Randomized Approximation for Max 3-SAT

Sorting problem revisited

Problem

- Input: a list L of n elements from a totally ordered universe
- Goal: rearrange them in ascending order

Examples

- $[2, 3, 1] \longrightarrow [1, 2, 3]$
- $[4, 2, 8, 5, 7] \longrightarrow [2, 4, 5, 7, 8]$
- ["s","o","r","t"] ⟶ ["o","r","s","t"]

Merge sort solves sorting in $\mathrm{O}(n \log n)$ time, but we study another algorithm

Merge sort requires $n/2$ extra spaces

Quick Sort

$\mathtt{qsort}(L)$

if $|L| \leq 1$ **then Return** L;
Let x be the first element of L;
$A \leftarrow [e \in L \mid e < x]$, $B \leftarrow [e \in L \mid e = x]$, and $C \leftarrow [e \in L \mid e > x]$;
Return $\mathtt{qsort}(A) + B + \mathtt{qsort}(C)$;

Quick sort works in-place

- Optimistic case: $|A|, |B| \approx |L|/2$
  $T(n) = 2T(n/2) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n \log n)$
- Worst case: $|A| = 0$ (L is sorted in descending order)
  $T(n) = T(n-1) + \mathrm{O}(n) \longrightarrow T(n) = \mathrm{O}(n^2)$

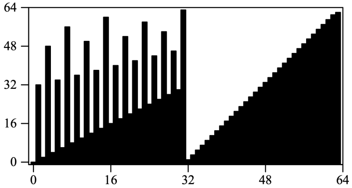
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Median-of-three Quick Sort

tqsort(L)

if $|L| \leq 1$ **then Return** L;
Let x be the median of the first, middle, last elements of L;
$A \leftarrow [e \in L \mid e < x]$, $B \leftarrow [e \in L \mid e = x]$, and $C \leftarrow [e \in L \mid e > x]$;
Return tqsort(A) $+ B +$ tqsort(C);

- a better estimate of the optimal pivot (the true median)
- but still requires $\mathrm{O}(n^2)$ time in the worst case



Doug McIlroy: "A Killer Adversary for Quicksort", 1999

Randomized Quick Sort

rqsort(L)

if $|L| \le 1$ **then Return** L;
Choose an element x uniformly at random from L;
$A \leftarrow [e \in L \mid e < x]$, $B \leftarrow [e \in L \mid e = x]$, and $C \leftarrow [e \in L \mid e > x]$;
Return $\texttt{rqsort}(A) + B + \texttt{rqsort}(C)$;

- Let a_i be the ith smallest element in L
- a_i and a_j $(i < j)$ are compared only if one of them is selected as x first in $a_i, a_{i+1}, \dots, a_j$ → they are compared with probability $\frac{2}{j-i+1}$
- The expected number of comparisons (≈ computational complexity) is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \le 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} = \mathrm{O}(n \log n)$$

$\sum_{k=1}^{n} \frac{1}{k} \le 1 + \int_1^n \frac{1}{x}\,dx = 1 + \log n$

Outline

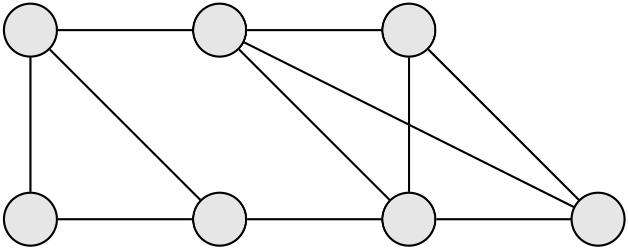
(Global) Min-cut Problem

Problem

- Input: connected undirected graph $G = (V, E)$
- Goal: find a partition (S, T) of V with minimum capacity $\text{cap}(S)$

$|\{e = \{u, v\} \in E : u \in S, v \notin S\}|$

Example



This problem can be solved by using s–t cut algorithm $|V| - 1$ times. But we study a simpler algorithm.

(Global) Min-cut Problem

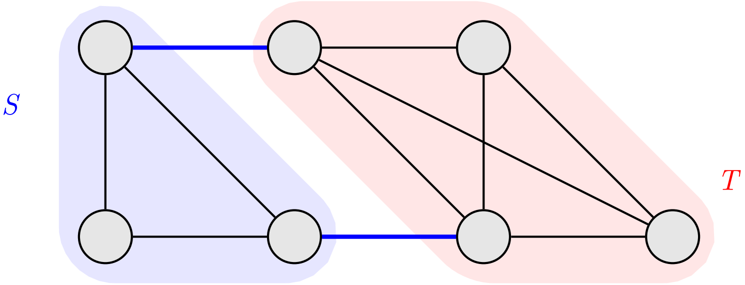
Problem

- Input: connected undirected graph $G = (V, E)$
- Goal: find a partition (S, T) of V with minimum capacity $\mathrm{cap}(S)$

$|\{e = \{u, v\} \in E : u \in S, v \notin S\}|$

Example

$\mathrm{cap}(S) = 2$



This problem can be solved by using s–t cut algorithm $|V| - 1$ times. But we study a simpler algorithm.

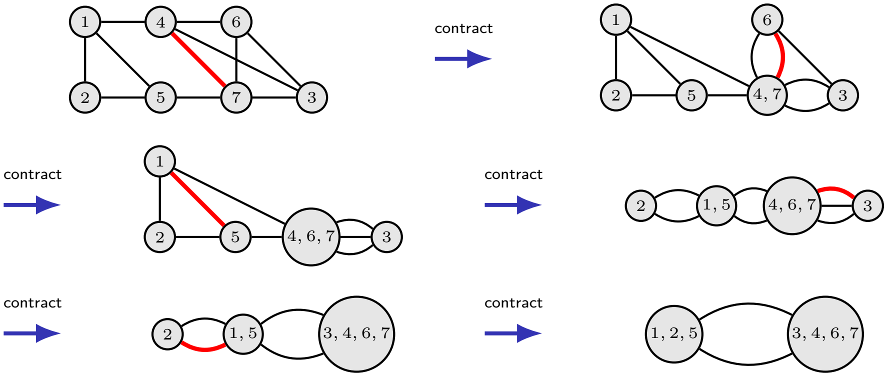
Karger's algorithm

while $|V| > 2$ **do**

  Pick an edge uniformly at random and contract it;

  Remove self-loops;

Return the partition corresponding to the remaining two vertices;



Analysis

Notations:

- C: the set of minimum cut edges
- $k := |C|$ and $n := |V|$
- $\mathcal{E}_i$: the event of not picking an edge of C at ith step

Observations:

- At each ith step,
  - degree of any vertices is at least k ⟶ $\#\text{edges} \geq k \cdot \frac{n-i+1}{2}$
  - $\Pr[\mathcal{E}_i \mid \mathcal{E}_1, \ldots, \mathcal{E}_{i-1}] \geq 1 - \frac{k}{\frac{k(n-i+1)}{2}} = 1 - \frac{2}{n-i+1}$
- no edge of C is ever picked with probability at least

$$\Pr\left[\bigcap_{i=1}^{n-2} \mathcal{E}_i\right] \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)} > \frac{2}{n^2}$$

Amplifying the success probability

- Karger's algorithm succeeds with probability $2/n^2$
- By running $\frac{n^2}{2}\log\frac{1}{\epsilon}$ times, the success probability is at least

$$1-\left(1-\frac{2}{n^2}\right)^{\frac{n^2}{2}\log\frac{1}{\epsilon}} \geq 1-\left(\frac{1}{e}\right)^{\log\frac{1}{\epsilon}} = 1-\epsilon,$$

where the inequality holds by $(1-x)^x \leq 1/e$ $(\forall x > 0)$

Outline

Verifying Matrix Multiplication

Problem

- Input: $n \times n$ matrices A, B, and C
- Goal: check whether $AB = C$ or not

Naive algorithm: compute $D = AB$ and check if $D = C$ ($\mathrm{O}(n^{2.372})$ time)

Can we do better by randomization?

Examples

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 1 \end{pmatrix},\ B = \begin{pmatrix} 4 & 2 \\ -1 & 3 \end{pmatrix},\ C = \begin{pmatrix} 3 & 19 \\ 11 & 9 \end{pmatrix}$$

Freivald's Algorithm

Pick $r \in \{0,1\}^n$ where each r_i is independent and uniform over $\{0,1\}$;
Return YES if $ABr = Cr$ and NO otherwise;

Running time: $\mathrm{O}(n^2)$

Theorem

The above algorithm outputs

- YES with probability 1 if $AB = C$
- YES with probability at most $1/2$ if $AB \neq C$

Proof: If $(AB)_{ij} \neq C_{ij}$ for some i, j, then $ABr^{(0)} \neq Cr^{(0)}$ or $ABr^{(1)} \neq Cr^{(1)}$ for $r^{(x)} = (r_1, \ldots, r_{i-1}, x, r_{i+1}, \ldots, r_n)$

Repeating $\log \frac{1}{\epsilon}$ times gives an $\mathrm{O}(n^2 \log \frac{1}{\epsilon})$ time algorithm with error $\leq \epsilon$

Polynomial Identity Testing

Problem

- Input: a polynomial $p(x_1, \dots, x_n)$ of degree at most d
- Goal: check whether $p(x_1, \dots, x_n) \equiv 0$ or not

Example

- $d = 2$, $p(x, y) = x^2 - xy$ ⟶ NO
- $d = 3$, $p(x, y) = (x + 2y)^2(x - y) - x^2(x + 3y) + 4y^3$ ⟶ YES
- $d = n^2$, $p(x_{11}, \dots, x_{nn}) = \det(A) = \sum_{\sigma \in \mathcal{S}_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} x_{i\sigma(i)}$

If $n = 1$, it is sufficient to check $p(0) = p(1) = \cdots = p(d) = 0$ or not

since any nonzero polynomial of degree d has at most d real roots by the fundamental theorem of algebra

What if $n > 1$? Now, $p(x, y) = x^2 - y$ has infinitely many roots.

Algorithm

Let $S \subseteq \mathbb{R}$ be any set of size $2d$;
Pick $\alpha_1, \ldots, \alpha_n$ independently and uniformly at random from S;
Return YES if $p(\alpha_1, \ldots, \alpha_n) = 0$ and NO otherwise;

Schwartz–Zippel Lemma

If p is a nonzero polynomial of degree d and $S \subseteq \mathbb{R}$, then

$$\Pr_{\alpha_1,\ldots,\alpha_n \overset{\text{i.i.d.}}{\sim} \mathrm{U}(S)} [p(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}$$

This can be proved by induction on n (see, e.g., [Motwani and Raghavan: Randomized Algorithms])

Theorem

The above algorithm outputs

- YES with probability 1 if $p \equiv 0$
- YES with probability at most $1/2$ if $p \not\equiv 0$

Outline

Max 3-SAT

Problem

- Input: a CNF formula Φ where each clause contains exactly 3 literals
- Goal: find a truth assignment that satisfies as many clauses as possible

Examples

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

⟶ 3 clauses are satisfiable by setting $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$

Algorithm

set each variable independently to 0 or 1 with probability $\frac{1}{2}$;
Return the assignment;

Proposition

The above algorithm is a $\frac{7}{8}$-approximation in expectation.

- Each clause is satisfied with probability $1 - \left(\frac{1}{2}\right)^3 = \frac{7}{8}$.
- The expected number of satisfied clauses is $\frac{7}{8}|\Phi| \geq \frac{7}{8} \cdot \mathrm{OPT}$.

Corollary

There always exists a truth assignment that satisfies at least $\frac{7}{8}|\Phi|$ clauses.

Can we obtain such a solution?

→ Yes, by repeatedly applying the algorithm.

Repetition

while *True* **do**
  set each variable independently to 0 or 1 with probability $\frac{1}{2}$ each;
  If the assignment satisfies θ clauses **Return** the assignment;

$\lceil \frac{7}{8} \cdot |\Phi| \rceil$

Lemma

For a series of independent trials with success probability p,
the expected number of trials until the first success is $1/p$.

Proof

- Let N be the number of trials until the first success
- $\Pr[N \geq j] = (1-p)^{j-1}$
- $\mathbb{E}[N] = \sum_{j=1}^{\infty} \Pr[N \geq j] = \sum_{j=1}^{\infty} (1-p)^{j-1} = \frac{1}{1-(1-p)} = \frac{1}{p}$

Repetition

while *True* **do**
  set each variable independently to 0 or 1 with probability $\frac{1}{2}$ each;
  If the assignment satisfies θ clauses **Return** the assignment;

$\lceil \frac{7}{8} \cdot |\Phi| \rceil$

Let p_j be the probability that a random assignment satisfies exactly j clauses

- success probability $p := \sum_{j \geq \theta} p_j$
- $\mathbb{E}[\#\text{satisfaction}]$ is $\frac{7}{8} \cdot |\Phi| = \sum_{j=0}^{|\Phi|} j \cdot p_j = \sum_{j<\theta} j \cdot p_j + \sum_{j \geq \theta} j \cdot p_j$
  - $\sum_{j \geq \theta} j \cdot p_j \leq |\Phi| \cdot \sum_{j \geq \theta} p_j = |\Phi| \cdot p$
  - $\sum_{j<\theta} j \cdot p_j \leq (\theta - 1) \cdot \sum_{j<\theta} p_j = (\theta - 1) \cdot (1 - p)$
- Hence, $p \geq \frac{\frac{7}{8}|\Phi| - (\theta-1)}{|\Phi| - (\theta-1)} \geq \frac{\frac{7}{8}|\Phi| - (\frac{7}{8}|\Phi| - \frac{1}{8})}{|\Phi|} = \frac{1}{8|\Phi|}$

→ The expected number of trials is at most $8|\Phi|$.