Success Amplification and Random Sampling

Faranak Tohidi

Department of Computer Science
Yazd University

April 25, 2016

Outline

Success Amplification and Random sampling

- amplification of success probability by repeating runs on the same input and random sampling
- amplification does not only increase the success probability, but directly stamps the process of the algorithm design
- prefers to repeat only some computation parts or to repeat different parts differently many times.
- more attention to computation parts in which the probability of making mistakes is greater
- designing efficient randomized algorithms solving problems for which no deterministic polynomial-time algorithm has up to now been discovered

Success Amplification and Random sampling

- amplification of success probability by repeating runs on the same input and random sampling
- amplification does not only increase the success probability, but directly stamps the process of the algorithm design
- prefers to repeat only some computation parts or to repeat different parts differently many times.
- more attention to computation parts in which the probability of making mistakes is greater
- designing efficient randomized algorithms solving problems for which no deterministic polynomial-time algorithm has up to now been discovered

Success Amplification and Random sampling

- amplification of success probability by repeating runs on the same input and random sampling
- amplification does not only increase the success probability, but directly stamps the process of the algorithm design
- prefers to repeat only some computation parts or to repeat different parts differently many times.
- more attention to computation parts in which the probability of making mistakes is greater
- designing efficient randomized algorithms solving problems for which no deterministic polynomial-time algorithm has up to now been discovered

Success Amplification and Random sampling

- amplification of success probability by repeating runs on the same input and random sampling
- amplification does not only increase the success probability, but directly stamps the process of the algorithm design
- prefers to repeat only some computation parts or to repeat different parts differently many times.
- more attention to computation parts in which the probability of making mistakes is greater
- designing efficient randomized algorithms solving problems for which no deterministic polynomial-time algorithm has up to now been discovered

Outline

Efficient Amplification by repeating Critical Computation Parts

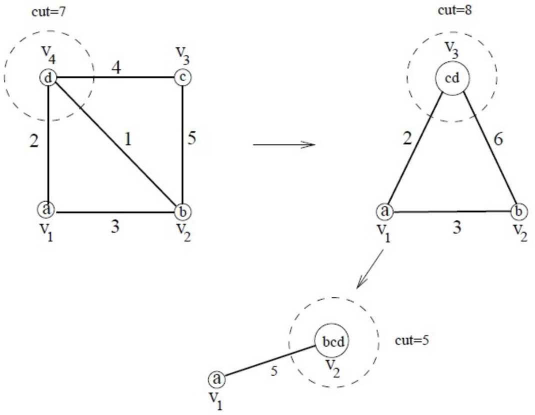
Introducing the method of amplification of the success probability as a method for the design of randomized algorithms.

MIN-CUT

- Input: A multigraph **G = (V, E, c)**, where **C: E** $\rightarrow$ **N** -{**0**}
- Constraints: The set of all feasible solutions for G is the set **M(G) =**{**(V1, V2)** |**V1**$\cup$**V2=V, V1**$\cap$**V2=0**}
- Costs: For every cut **(V1, V2)**ϵ**M(G), cost((V1, V2),G) =** $\sum$**c(e)** {**i.e., cost((V1, V2),G)** is equal to the number of edges between V1 and V2}
- Goal: minimum

MIN-CUT (deterministic algorothm)

- The best known deterministic algorithm for MIN-CUT runs in time $\mathbf{O}(|V|.|E|.log(\frac{|V|^2}{|E|}))$
- in the worst case, is in $\mathrm{O(n^3)} forn = |V|$.



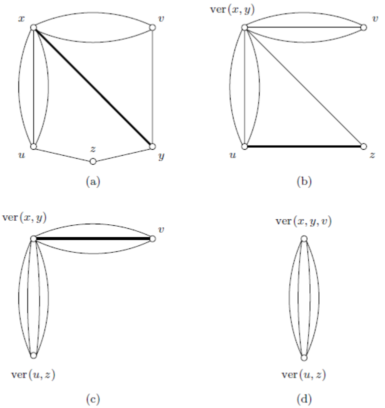
Contraction

Contract(G, e)

G = (V,E) and an edge e = $\{x, y\} \in$ **E**

1. the vertices x and y are replaced by a new vertex ver(x, y),
2. the multi-edge e =$\{x, y\}$ is removed (contracted).
3. each edge $\{r, s\}$ with an $r \in \{x, y\}$ and $s \in \{x, y\}$ is replaced by a new edge $\{ver(x, y), s\}$
4. all remaining parts of G remain unchanged.

MIN-CUT



1. the effect of contracting the edges in F⊆ E does not depend on the order of contractions.
2. the multigraph in Figure 5.1(d) with a cost of 4.

Outline

Randomized Algorithm

One contracts randomly chosen edges until one gets a multigraph with exactly two vertices ver(V1) and ver(V2).

Algorithm CONTRACTION

- Input: A connected multigraph $G = (V, E, c)$
- Step 1: Set label $(v) := \{ v \}$ for every vertex $v \in V$.
- Step 2: while G has more than two vertices do

  begin

  choose an edge $e = \{x, y\} \in E(G)$;

  $G := \text{Contract}(G, e)$;

  Set label $(z) :=$ label $(x) \cup$ label (y)

  for the new vertex $z = \text{ver}(x, y)$;

  end
- Step 3: if $G = (\{u, v\}, E(G))$ for a multiset $E(G)$ then
- output (label (u) , label (v)) and cost $= |E(G)|$

Algorithm CONTRACTION

Theorem 5.2.1

The algorithm CONTRACTION is a randomized polynomial-time algorithm that computes a minimal cut of a given multigraph G of n vertices with probability at least $\dfrac{2}{n.(n-1)}$

Theorem 5.2.1

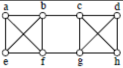
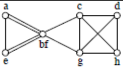
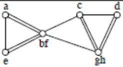
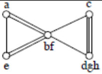
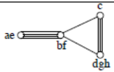
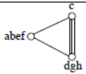
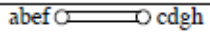
- The algorithm consists of **n-2** contractions.
- time complexity of CONTRACTION is in $O(n^2)$
- Let G = (V, E, c) be a multigraph,

  and let $(C - min) = (V1, V2)$ be a minimal cut of G with cost $(C - min) = k$ for a natural number k.
- Every vertex of G has a degree of at least k.

  G has at least $\frac{nk}{2}$ edges, i.e., $|E(G)| \geq \frac{nk}{2}$
- Important observation is that

the algorithm CONTRACTION computes (C_{min}) if and only if no edge from E(C_{min}) has been contracted.

CONTRACTION

	14 edges to choose from Pick $b - f$ (probability 1/14)
	13 edges to choose from Pick $g - h$ (probability 1/13)
	12 edges to choose from Pick $d - gh$ (probability 1/6)
	10 edges to choose from Pick $a - e$ (probability 1/10)
	9 edges to choose from Pick $ab - ef$ (probability 4/9)
	5 edges to choose from Pick $c - dgh$ (probability 3/5)
	Done: just two nodes remain

Theorem 5.2.1

Probability

- Let $S_{Con,G}$,G be the set of all possible computations of the algorithm CONTRACTION on G.
- $Event_i$ = { all computations from $S_{Con,G}$ in which no edge of $E(C_{min})$ is contracted in the i-th contraction step }

  for i = 1, 2, . . ., n-2.
- The event that (C_{min}) is the output of the algorithm is exactly the event:

$$\bigcap_{i=1}^{n-2} Event_i$$

$$\operatorname{Prob}\left(\bigcap_{i=1}^{n-2}\text{Event}_i\right) = \operatorname{Prob}(\text{Event}_1)\cdot\operatorname{Prob}(\text{Event}_2 \mid \text{Event}_1)$$
$$\cdot\operatorname{Prob}(\text{Event}_3 \mid \text{Event}_1 \cap \text{Event}_2)\cdot\ldots$$
$$\cdot\operatorname{Prob}\left(\text{Event}_{n-2} \,\middle|\, \bigcap_{j=1}^{n-3}\text{Event}_j\right)$$

To prove Theorem 5.2.1, we have to estimate lower bounds on

$$\operatorname{Prob}\left(\text{Event}_i \,\middle|\, \bigcap_{j=1}^{i-1}\text{Event}_j\right)$$

for $i = 1, \ldots, n-2$.

- Since G has at least $\frac{nk}{2}$ edges and the algorithm makes a random choice for edge contraction,

$$\begin{aligned}\operatorname{Prob}(\text{Event}_1) &= \frac{|E| - |E(C_{\min})|}{|E|} \\ &= 1 - \frac{k}{|E|} \\ &\geq 1 - \frac{k}{\frac{k\cdot n}{2}} = 1 - \frac{2}{n}.\end{aligned}$$

contraction

- In general, the multigraph G/Fi created after $i-1$ random contractions has exactly $n-i+1$ vertices. If

$$F_i \cap E(C_{\min}) = \emptyset \text{ (i.e., } \bigcap_{j=1}^{i-1} \text{Event}_j \text{ happens)}$$

- Every vertex in G/Fi has still to have a degree of at least k, and so G/Fi has at least

$$\frac{k.(n-i+1)}{2} \text{ edges.}$$

Contraction

- Therefore

$$\text{Prob}\left(\text{Event}_i \;\middle|\; \bigcap_{j=1}^{i-1} \text{Event}_j\right) \geq \frac{|E(G/F_i) - E(C_{\min})|}{|E(G/F_i)|}$$
$$\geq 1 - \frac{k}{\frac{k\cdot(n-i+1)}{2}}$$
$$= 1 - \frac{2}{(n-i+1)}$$

- for i $= 2, \ldots, n-1$.

$$\text{Prob}\left(\bigcap_{j=1}^{n-2} \text{Event}_j\right) \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right)$$
$$= \prod_{l=n}^{3}\left(\frac{l-2}{l}\right)$$
$$= \frac{2}{n\cdot(n-1)} = \frac{1}{\binom{n}{2}}.$$

- Theorem 5.2.1 assures that the probability of discovering a particular minimal cut in one run is at least:

$$\frac{2}{n.(n-1)} \succ \frac{2}{n^2}$$

- One does not obtain a minimal cut with a probability of at most:

$$(1-\frac{2}{n^2})^{n^2/2} \prec \frac{1}{e}$$

- Hence, the complementary probability of computing a minimal cut by $n^2/2$ runs of the algorithm is at least:

$$1-\frac{1}{e}$$

- The complexity of the algorithm $CONTRACTION_{n^2/2}$ is in $O(n^4)$.

- The probability of contracting an edge from Cmin grows with the number of contractions executed.

$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \frac{2}{n-4}, \dots$$

- The created multigraph G/F, is small enough to be searched for a minimal cut in a deterministic way.

Outline

Algorithm DETRAN(l)

The size of G/F will still remain a free parameter of the algorithm.

Let $L : N \to N$ be a monotonic function such that $1 < l(n) < n$ for every n $\in N$.

- Input: A multigraph G = (V, E, c) of n edges, n $n \in N$, $n \geq 3$.
- Step 1: Perform the algorithm CONTRACTION on G in order to get a multigraph G/F of l(n) vertices.
- Step 2: Apply the best known deterministic algorithm on G/F to compute an optimal cut D of G/F.
- Output: D

DETRAN(l)

- Analyze the influence of the exchange of CONTRACTION for DETRAN(l) on:
  1. (i) the amplification of the success probability
  2. (ii) the increase of the complexity.
- Consider l as a free parameter, i.e., the result of the analysis depends on l.

Lemma 5.2.3

Let $L : N \to N$ be a monotonic function such that $1 < l(n) < n$ for every n $\in N$.

- The algorithm DETRAN(l) works in time:

$$O(n^2 + (l(n))^3)$$

- and it finds an optimal solution with probability at least:

$$\frac{\binom{l(n)}{2}}{\binom{n}{2}}$$

Lemma 5.2.3

Complexity of DETRAN(l)

- In step 1, $(n - l)(n)$ contractions are performed, and each contraction can be executed in time O(n).
  Hence, step 1 can be executed in:
  $$O((n - l(n)).n) = O(n^2)$$
- Deterministic algorithm can compute an optimal cut of G/F in time:
  $$((l(n))^3)$$
- Altogether, the complexity of DETRAN(l) is in:
  $$O(n^2 + (l(n))^3)$$

Success probability of DETRAN(l)

- Let (C_{min}) be a minimal cut of G.
- lower bound on the probability of having (C_{min}) in the multigraph G/F after executing step 1:

$$\bigcap_{i=1}^{n-l(n)} \mathrm{Event}_i$$

$$\begin{aligned}
\mathrm{Prob}\left(\bigcap_{i=1}^{n-l(n)} \mathrm{Event}_i\right) &\geq \prod_{i=1}^{n-l(n)} \left(1 - \frac{2}{n-i+1}\right) \\
&= \frac{\prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right)}{\prod_{j=n-l(n)+1}^{n-2}\left(1-\frac{2}{n-j+1}\right)} \\
&= \frac{\frac{1}{\binom{n}{2}}}{\frac{1}{\binom{l(n)}{2}}} = \frac{\binom{l(n)}{2}}{\binom{n}{2}}.
\end{aligned}$$

Time complexity and Success probability of DETRAN(l)

- Since
  $$\frac{n^2}{(l(n))^2} \geq \frac{\binom{n}{2}}{\binom{l(n)}{2}}$$
- $\dfrac{n^2}{(l(n))^2}$ independent runs of DETRAN(l) provide a randomized algorithm that works in time:
  $$O\left((n^2 + (l(n))^3) \cdot \frac{n^2}{(l(n))^2}\right) = O\left(\frac{n^4}{(l(n))^2} + n^2 \cdot l(n)\right)$$
- Computes a minimal cut of G with probability at least:
  $1 - \dfrac{1}{e}$
- The best possible choice of l with respect to the time complexity is:
  $$l(n) = \lfloor n^{2/3} \rfloor$$

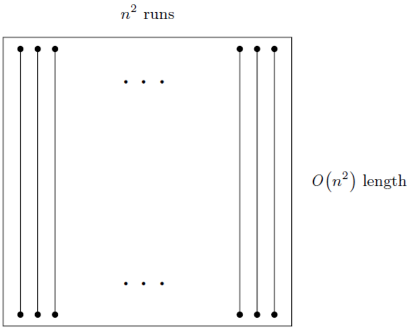
DETRAN(l)

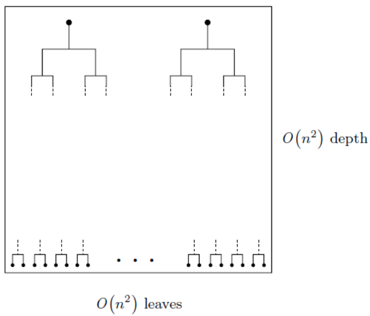
Theorem 5.2.4.

- The algorithm DETRAN($\lfloor n^{2/3} \rfloor_{n^2/\lfloor n^{2/3} \rfloor}$) works in time:
$$O(n^{8/3})$$
- Computes a minimal cut with probability at least:
$$1 - e^{-1}$$

Outline

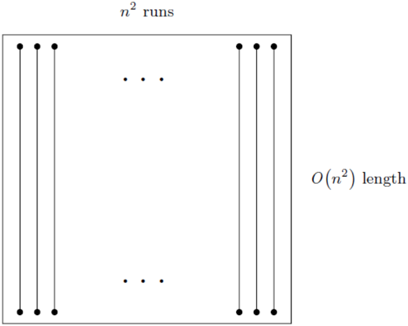
RepTree

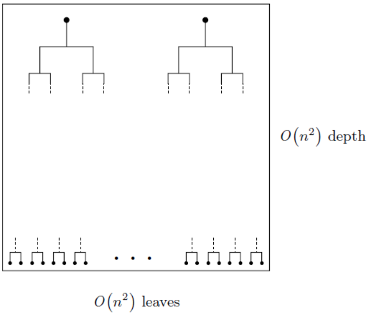




$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \frac{2}{n-4}, \ldots, \frac{2}{3}$$

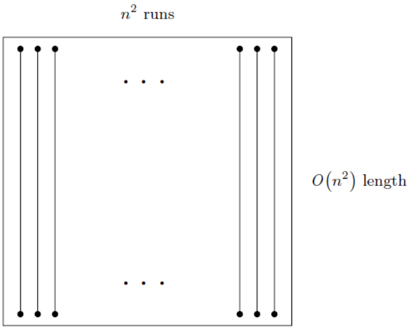
RepTree

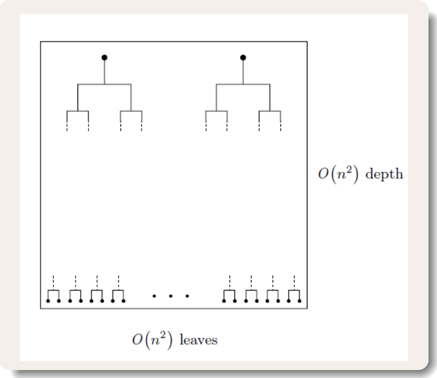




$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \frac{2}{n-4}, \ldots, \frac{2}{3}$$

RepTree





$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \frac{2}{n-4}, \ldots, \frac{2}{3}$$

Algorithm REPTREE(G)

- Input: A multigraph G = (V, E, c), $|V| = n, n \in N, n \geq 3$.
  Procedure:
- if $n \leq 6$ then:
  compute a minimal cut deterministically
  else
  begin
- h:= $\lceil 1 + \frac{n}{\sqrt{2}} \rceil$
  Perform two independent runs of CONTRACTION in order to get two multigraphs G/F1 and G/F2 of size h;
  REPTREE(G/F1);
  REPTREE(G/F2)
  end
- output the smaller of the two cuts computed by REPTREE(G/F1) and REPTREE(G/F2)

Algorithm REPTREE

Theorem 5.2.5.

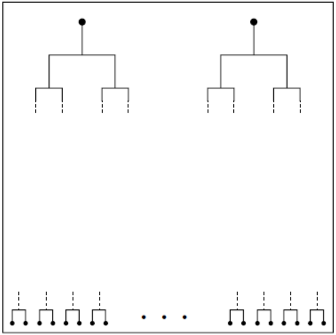
The algorithm REPTREE works in time $O(n^2.logn)$ and finds a minimal cut with a probability of at least:

$$\frac{1}{\Omega(\log_2^n)}$$

Reptree

Time complexity

1. Depths = number of recursion calls of Reptree.
2. Size is reduced by: $\frac{1}{\sqrt{2}}$
3. Number of recursion calls:
$$\log_{\sqrt{2}}^{n} \in O(\log_{2}^{n})$$
4. Number of leaves:
$$2^{\left(\log_{\sqrt{2}}^{(n-2)}\right)} \in O(n^{2})$$



The depths of the binary trees correspond to the number of recursion calls of REPTREE.

Reptree

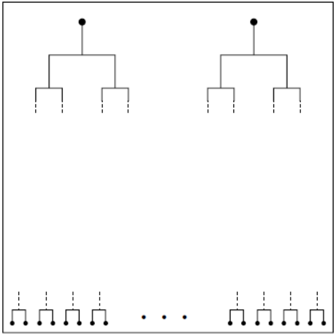
Time complexity

1. Depths = number of recursion calls of Reptree.
2. Size is reduced by: $\frac{1}{\sqrt{2}}$
3. Number of recursion calls:

$$\log^{n}_{\sqrt{2}} \in O(\log^{n}_{2})$$

4. Number of leaves:

$$2^{(\log^{(n-2)}_{\sqrt{2}})} \in O(n^{2})$$



The depths of the binary trees correspond to the number of recursion calls of REPTREE.

Reptree

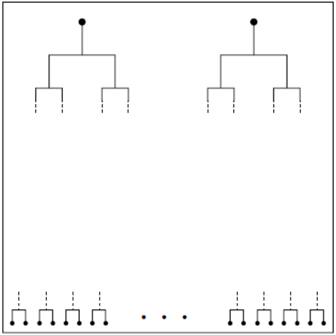
Time complexity

1. Depths = number of recursion calls of Reptree.
2. Size is reduced by: $\frac{1}{\sqrt{2}}$
3. Number of recursion calls:

$$\log_{\sqrt{2}}^{n} \in O(\log_{2}^{n})$$

4. Number of leaves:

$$2^{\left(\log_{\sqrt{2}}^{(n-2)}\right)} \in O(n^2)$$



The depths of the binary trees correspond to the number of recursion calls of REPTREE.

Reptree

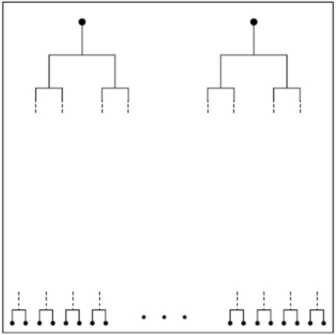
Time complexity

1. Depths = number of recursion calls of Reptree.
2. Size is reduced by: $\frac{1}{\sqrt{2}}$
3. Number of recursion calls:

$$\log_{\sqrt{2}}^{n} \in O(\log_{2}^{n})$$

4. Number of leaves:

$$2^{\left(\log_{\sqrt{2}}^{(n-2)}\right)} \in O(n^2)$$



The depths of the binary trees correspond to the number of recursion calls of REPTREE.

Time Complexity Algorithm REPTREE

- TimeREPTREE(n) $\in O(1)$ for $n \leq 6$
- TimeREPTREE(n) $= 2$ TimeREPTREE $(\lceil 1 + \frac{n}{\sqrt{2}} \rceil) + O(n^2)$
- TimeREPTREE(n) $= \Theta(n^2 . \log_2^n)$

Lower bound on the Success Probability

- Let p_l be the probability that G/Fi (i = 1, 2) still contains C_{min}.

$$p_l \geq \frac{\binom{\lceil 1+\frac{l}{\sqrt{2}} \rceil}{2}}{\binom{l}{2}} = \frac{\lceil 1+\frac{l}{\sqrt{2}} \rceil \cdot \left(\lceil 1+\frac{l}{\sqrt{2}} \rceil - 1 \right)}{l \cdot (l-1)} \geq \frac{1}{2}$$

- Probability that RepTree finds C_{min} for i=1,2.

$$p_l \cdot \mathrm{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil \right)$$

- Probability that RepTree does not find C_{min}.
  G/F has contained C_{min}.

$$\left(1 - p_l \cdot \mathrm{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil \right) \right)^2$$

Probability of RepTree

- So the following recurrence for Prob(n) is obtained.

$$\begin{aligned}
&\mathrm{Prob}(2) = 1, \text{ and} \\
&\mathrm{Prob}(l) \geq 1 - \left(1 - p_l \cdot \mathrm{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil\right)\right)^2 \\
&\qquad\quad \geq 1 - \left(1 - \frac{1}{2} \cdot \mathrm{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil\right)\right)^2
\end{aligned}$$

- Prob satisfying the recurrence is in:

$$\Theta \frac{1}{\log_2^n}$$

- $O(\log_2^n)$ repetitions of the algorithm REPTREE are sufficient in order to compute a minimal cut with a constant probability.

Algorithm RepTree

Complexity

- $O((\log_2^n)^2)$ repetitions suffice to reduce the non-success probability to a function tending to 0 with growing n and one can consider this algorithm applicable.
- So, The complexity of $REPTREE_{(\log_2^n)^2}$ is in:

$$O(n^2.(\log_{2\ n})^3)$$

Algorithm RepTree

complexity Comparison

- The complexity of $REPTREE_{(\log_2^n)^2}$ is in:

$$O(n^2.(\log_{2\ n})^3)$$

which is substantially better than the complexity

$$O(n^3)$$

of the best deterministic algorithm and the complexity

$$O(n^{8/3})$$

of the randomized algorithm DETRAN$(\lfloor n^{2/3} \rfloor_{n^2/\lfloor n^{2/3} \rfloor})$.

Repeated Random Sampling and Satisfiability

Combination amplification and random sampling

- Combine amplification and random sampling with local search in order to design:
- A randomized algorithm that can solve the 3-satisfiability problem (3SAT).

Outline

SCHONING

Algorithm SCHONING

- Input: A formula F in 3CNF over n Boolean variables.
- Step 1: NUMBER := 0; ATMOST :=X; FOUND := FALSE;
- Step 2: while NUMBER < ATMOST and FOUND = FALSE do

  begin

  NUMBER := NUMBER + 1; Generate at random an assignment $\alpha \in \{0, 1\}^n$;

  if F is satisfied by then FOUND := TRUE;

SCHONING

Algorithm SCHONING Cont

- M := 0;
- while $M < 3.n$ and FOUND = FALSE do

  begin

  M := M + 1; Find a clause C that is not satisfied by α.

  Pick one of the literals of C at random, and flip the value of its variable.

  if α satisfies F then

  FOUND := TRUE;

  end

  end
- Step 3: if FOUND = TRUE then

  output: *F is satisfiable.*

  else output: *F is not satisfiable.*

Algorithm SCHONING

SCHONING is a 1MC algorithm for 3SAT

1. Let F be not satisfiable. Then, the algorithm SCHONING does not find any assignment that satisfies F and outputs the correct answer:

   F is not satisfiable with certainty.

2. Let F be satisfiable.

   To prove a lower bound on the success probability of SCHONING; we analyze the probability of finding a certain assignment α^* that satisfies F. Let α and β be two assignments.

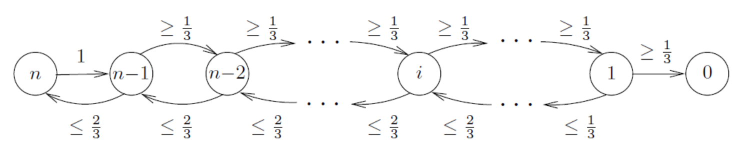
   $Dist(\alpha, \beta)$ between α and β as the number of bits in which they differ.

$$Class(j) = \{\beta \in \{0,1\}^n | Dist(\alpha^*, \beta) = j\}.$$

$$|Class(j)| = \binom{n}{j}$$

SCHONING

Algorithm SCHONING



- The probability of moving from vertex j to vertex $j-1$ in one local search step is at least $\frac{1}{3}$

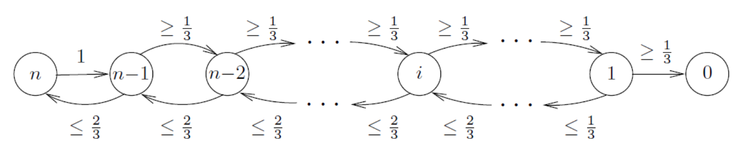
$$Class(j) = \{\beta \in \{0,1\}^n | Dist(\alpha^*, \beta) = j\}.$$

$$|Class(j)| = \binom{n}{j}$$

The probability $q_{j,i}$ that the algorithm SCHONING starting in the Class (j) (in the vertex j) reaches α^*(the vertex 0) in exactly $j+2i$ local steps.

SCHONING

Algorithm SCHONING



- j +i steps toward α^* and i steps in the opposite direction (toward the vertex n). Since:

$$j + 2i \leqslant 3.n$$

- The movement of the algorithm during the j+2i steps on the graph by a word (string) in $\{+,-\}^{j+2i}$.
- The number of words over$\{+,-\}$of length j + 2i with exactly i symbols is

$$\binom{j+2.i}{i}$$

- Only those words correspond to possible runs for which every suffix contains at least as many + symbols as - symbols.

$$\frac{j}{j+2i} \cdot \binom{j+2.i}{i}$$

- Since the symbol + occurs with a probability of at least 1/3 and the symbol occurs with a probability of at most 2/3,

$$\mathrm{Prob}(\mathrm{Event}(w)) \geq \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^{i}$$

- So

$$q_{j,i} \geq \frac{1}{j+2i} \cdot \binom{j+2\cdot i}{i} \cdot \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^{i}$$

- q_j be the probability of reaching α^* from an $\alpha \in$ Class (j) in at most 3n local steps.

$$q_j \geq \sum_{i=0}^{j} q_{j,i}$$

$$\begin{aligned} q_j &\geq \sum_{i=0}^{j} \left[\frac{1}{j+2i} \cdot \binom{j+2\cdot i}{i} \cdot \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^{i} \right] \\ &> \frac{1}{3} \cdot \binom{3\cdot j}{j} \cdot \left(\frac{1}{3}\right)^{2\cdot j} \cdot \left(\frac{2}{3}\right)^{j} \end{aligned}$$

- Stirling formula

$$r! = \sqrt{2\pi r}\left(\frac{r}{e}\right)^r \cdot \left(1 + \frac{1}{12r} + O\left(\frac{1}{r^2}\right)\right) \sim \sqrt{2\pi r} \cdot \left(\frac{r}{e}\right)^r$$

- Inserting the Stirling formula

$$\begin{aligned} q_j &\geq \frac{1}{3} \cdot \frac{(3\cdot j)!}{(2\cdot j)! \cdot j!} \cdot \left(\frac{1}{3}\right)^{2\cdot j} \cdot \left(\frac{2}{3}\right)^{j} \\ &\geq \frac{1}{3} \cdot \frac{\sqrt{2\pi \cdot 3j} \cdot \left(\frac{3\cdot j}{e}\right)^{3\cdot j}}{\sqrt{2\pi \cdot 2j} \cdot \left(\frac{2\cdot j}{e}\right)^{2\cdot j} \cdot \sqrt{2\pi j} \cdot \left(\frac{j}{e}\right)^{j}} \cdot \left(\frac{1}{3}\right)^{2\cdot j} \cdot \left(\frac{2}{3}\right)^{j} \\ &= \frac{1}{3} \cdot \frac{\sqrt{3}}{2 \cdot \sqrt{\pi j}} \cdot \frac{3^{3\cdot j}}{2^{2\cdot j}} \cdot \left(\frac{1}{3}\right)^{2\cdot j} \cdot \left(\frac{2}{3}\right)^{j} \\ &= \frac{1}{2 \cdot \sqrt{3\pi j}} \cdot \left(\frac{1}{2}\right)^{j}. \end{aligned}$$

- $p_j = \binom{n}{j}/2^n$, $q_j \geq \frac{1}{2.\sqrt{3\pi j}}.(\frac{1}{2})^j$

- Probability of SCHONING finds α^* by a random sample followed by a local search of at most 3n steps $\geqslant \sum_{j=0}^{n} p_j.q_j$

$$
\begin{aligned}
p &> \sum_{j=0}^{n} \left[\left(\frac{1}{2}\right)^n \cdot \binom{n}{j} \cdot \left(\frac{1}{2 \cdot \sqrt{3\pi j}} \cdot \left(\frac{1}{2}\right)^j \right) \right] \\
&\geq \frac{1}{2 \cdot \sqrt{3\pi n}} \cdot \left(\frac{1}{2}\right)^n \cdot \sum_{j=0}^{n} \left[\binom{n}{j} \cdot \left(\frac{1}{2}\right)^j \right] \\
&= \frac{1}{2 \cdot \sqrt{3\pi n}} \cdot \left(\frac{1}{2}\right)^n \cdot \left(1 + \frac{1}{2}\right)^n
\end{aligned}
$$

- $= \frac{1}{2.\sqrt{3\pi n}}.\left(\frac{3}{4}\right)^n = \tilde{p}$

- The probability that the algorithm does not find any assignment satisfying F by one random sample followed by the considered local search is at most:

$$1 - \tilde{p}$$

- The error probability after t independent attempts of the algorithm is at most:

$$(1 - \tilde{p})^t \leq e^{-\tilde{p}.t}$$

- Since : $\tilde{p} = \frac{1}{2.\sqrt{3\pi n}} . \left(\frac{3}{4}\right)^n$
- Taking t = ATMOST $= 20.\sqrt{3\pi n}. \left(\frac{4}{3}\right)^n$ and inserting it into $(1 - \tilde{p})^t \leq e^{-\tilde{p}.t}$:

$$Error_{SCHONING}(F) \leq (1 - \tilde{p})^t \leq e^{-10} < 5.10^{-5}$$

Thus, we have proved that the algorithm SCHONING is a one-sided-error Monte Carlo algorithm for 3SAT.

Algorithm SCHONING

Time Complexity

- The algorithm SCHONING is a 1MC algorithm for the 3SAT-problem that runs in time:

$$O(|F|.n^{3/2}.(\frac{4}{3})^n)$$

for any instance F of n variables.

Proof

- **Step 2**$\Rightarrow X = ATMOST = 20.\sqrt{3\pi n}.\left(\frac{4}{3}\right)^n$
- Each local search consists of at most $3n$ steps.
- Each step can be performed in time$O(|F|)$.
  So $\Longrightarrow$ Time Complexity of Algorithm SCHONING is:

$$O(|F|.n^{3/2}.(\frac{4}{3})^n)$$

Outline

Appendix

Lemma A.3.69

- Word (i, j) denote the set of all strings (words) over the alphabet$\{+, -\}$ of length $j + 2i > 0, j > 0$, that have the following properties:

  (i) The number of symbols 1 is exactly j +i (i.e., the number of 0s is exactly i)

  (ii) every suffix of the string contains more 1s than 0s.

Then: $|Word(i,j)| = \binom{j+2i}{i} . \frac{j}{j+2i}$

Lemma A.3.69

proof by induction

(i) We prove the claim for $n \leq 3$:

- Word (0, j):

$$\binom{j}{0}.\frac{j}{j} = 1$$

- Word (1, 1) $= \{011\}$

$$\binom{3}{1}.\frac{1}{3} = 1$$

(ii) Let n > 3.

- Word (i, j) beginning with the symbol 1 is exactly

$$|Word(i, j-1)| = \binom{j-1+2i}{i} . \frac{j-1}{j-1+2i}$$

- The number of words in Word (i, j) that start with 0 is:

$$|Word(i-1, j+1)| = \binom{j+1+2(i-1)}{i-1} . \frac{j+1}{j+1+2(i-1)}$$

- Now, we distinguish two cases, namely j = 1 and j > 1.

(ii).1 Let j = 1.

$$
\begin{aligned}
|\mathrm{Word}\,(i,1)\,| &= |\mathrm{Word}\,(i-1,2)\,| \\
&= \binom{2+2(i-1)}{i-1} \cdot \frac{2}{2(i-1)+2} \\
&= \binom{2i}{i-1} \cdot \frac{1}{i} = \frac{(2i)!}{(i-1)! \cdot (i+1)!} \cdot \frac{1}{i} \\
&= \frac{(2i)! \cdot (2i+1)}{i! \cdot (i+1)! \cdot (2i+1)} = \binom{2i+1}{i} \cdot \frac{1}{2i+1} \\
&= \binom{j+2i}{i} \cdot \frac{1}{j+2i}
\end{aligned}
$$

(ii).2 Let j> 1.

$$
\begin{aligned}
|\mathrm{Word}\,(i,j)\,| &= |\mathrm{Word}\,(i,j-1)\,| + |\mathrm{Word}\,(i-1,j+1)\,| \\
&= \binom{j-1+2i}{i} \cdot \frac{j-1}{2i+j-1} \\
&\quad + \binom{j+1+2(i-1)}{i-1} \cdot \frac{j+1}{2(i-1)+j+1} \\
&= \binom{j-1+2i}{j-1+i} \frac{j-1}{2i+j-1} \\
&\quad + \binom{j+2i-1}{i-1} \cdot \frac{j+1}{2i+j-1} \\
&= \frac{j}{2i+j} \cdot \binom{j+2i}{j+i} \\
&\quad \cdot \left(\frac{(j-1)(j+i)}{j \cdot (2i+j-1)} + \frac{(j+1) \cdot i}{j \cdot (2i+j-1)} \right) \\
&= \frac{j}{2i+j} \cdot \binom{j+2i}{j+i} \\
&= \binom{j+2i}{i} \cdot \frac{j}{j+2i}.
\end{aligned}
$$

Random Sampling and Generating Quadratic Nonresidues

Some Problem

1. One does not know any deterministic polynomial-time algorithm, and
2. There are no proofs presenting their NP-hardness.

Designing an efficient Las Vegas algorithm for a problem with properties (1) and (2).

Outline

quadratic nonresidue

- A quadratic residue in the field $\mathbb{Z}_p$ is any element $a \in \mathbb{Z}_p$ such that:
$$a \equiv x^2 mod \text{ p}$$
for an $x \in \mathbb{Z}_p$.
- A quadratic nonresidue is any number $b \in \mathbb{Z}_p$ such that:
$$d^2 \not\equiv b \text{ mod p}$$
for all $d \in \mathbb{Z}_p$.

modulus	quadratic residues	quadratic non-residues
2	0,1	(none)
3	0,1	2
4	0,1	2,3
5	0,1,4	2,3
6	0,1,3,4	2,5
7	0,1,2,4	3,5,6
8	0,1,4	2,3,5,6,7

Generation of a quadratic nonresidue

- Input: A prime $p > 2$.
- Output: A quadratic nonresidue modulo p.

Quadratic nonresidue

modulus	quadratic residues	quadratic non-residues
2	0,1	(none)
3	0,1	2
4	0,1	2,3
5	0,1,4	2,3
6	0,1,3,4	2,5
7	0,1,2,4	3,5,6
8	0,1,4	2,3,5,6,7

Theorem 5.4.14.Eulers Criterion

Let p, with p > 2, be a prime. For every a$\in \{1, 2, ..., p-1\}$,

- (i) if a is a quadratic residue modulo p, then
$$a^{(p-1)/2} \equiv 1 \text{ (mod p)}$$
- (ii) if a is a quadratic nonresidue modulo p, then
$$a^{(p-1)/2} \equiv p-1 \text{ (mod p)}$$

Theorem 5.4.15.

For every odd prime p, exactly half of the nonzero elements of $\mathbb{Z}_p$ are quadratic residues modulo p.

Solve the problem by random sampling

(A)

- For every prime p and every $a \in \mathbb{Z}_p$, one can efficiently decide (in a deterministic way) whether a is a quadratic residue or a quadratic nonresidue modulo p.

(B)

- For every prime p exactly half of the elements of $\mathbb{Z}_p - \{0\}$ are quadratic nonresidues, i.e., a random sample from $\{1, 2, \ldots, p-1\}$ provides a quadratic nonresidue with probability $1/2$.

Outline

Algorithm NQUAD

- Input: A prime p.
- Step 1: Choose uniformly an a $\in \{1, 2, \ldots, p-1\}$ at random.
- Step 2: Compute:

$$\mathrm{A} := a^{(p-1)/2} \bmod \mathrm{p}$$

  by the method of repeated squaring.
- Step 3:
  if $A = p - 1$ then
  output a
  else
  output *?*

The above proved claims (A) and (B) imply that
(i) NQUAD does not make any error, and
(ii) NQUAD finds a quadratic nonresidue with the probability $1/2$.

THE END