Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 8

2017-05-15

Summary of Singularity Analysis Excursion

World A	World B

World A

sequence of number T_n

$1, 1, \dots,$

- recurrence equations
- exact solution

$\downarrow$ if that does not seem to exist

- asymptotic approximation $(n \to \infty)$

World B

Generating Function

$$T(z) = \sum_{n \geq 0} T_n z^n$$

$$T(z) = \frac{1}{1-z}$$

- (functional) equation for $T(z)$

↳ exact solution

$\downarrow$ approximate $T(z)$

singular expansions $T(z) = f(z) + O((1-z)^\alpha)$

transfer theorem

$\rightsquigarrow$ interleaving gets rid of factor $r/q(k)$ in front of α^k $r(n) = O((\alpha-\varepsilon)^n)$

Possible Extensions

- (constant) coefficients $c_j \cdot T_{n-d_j}$ in recurrence
  $\rightsquigarrow$ different characteristic polynomial, same ideas
- *any* recurrence that leads to a representation of the generating function as a *singular expansion* around the dominant singularity.

$$\begin{aligned} f(z) &= c(1-z/z_0)^{-m} \;\pm\; \mathcal{O}((1-z/z_0)^{-m+1}) && (z \to z_0) \\ \rightsquigarrow [z^n]f(z) &= \frac{c}{(m-1)!} z_0^{-n} n^{m-1} \cdot \Big(1 \;\pm\; \mathcal{O}(n^{-1})\Big) && (n \to \infty) \end{aligned}$$

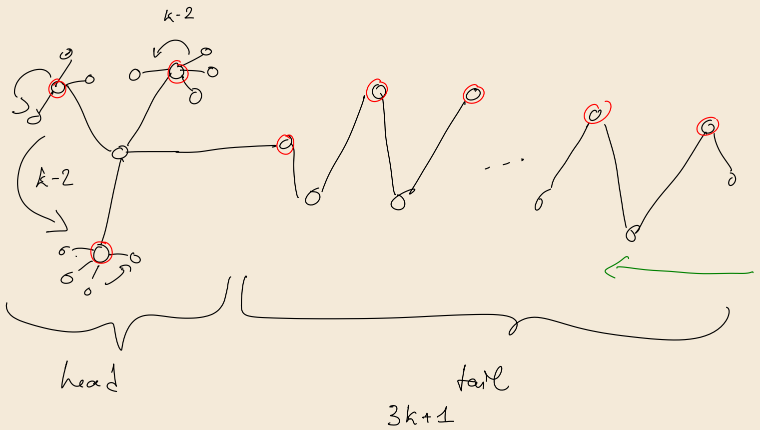
- other powers α in $1/(1-z)^{\alpha}$:

$$[z^n]\frac{1}{(1-\frac{z}{z_0})^{\alpha}} = \frac{z_0^{-n} n^{\alpha-1}}{\Gamma(\alpha)}\Big(1 \;\pm\; \mathcal{O}(n^{-1})\Big) \qquad (n \to \infty) \qquad \begin{array}{c} -\alpha \notin \mathbb{N}_0 \\ z_0 > 0 \end{array}$$

- much more! $\rightsquigarrow$ *analytic combinatorics*

Interleaving helps ... really!

$k \geq 2$



smallest VC for G_k has $\frac{5}{2}k - \frac{3}{2}$ vertices

($k-2$ in head
every other in tail)

(G_k, k) NO-instances

$$|G_k| = n = (k-2)(k-1) + 1$$
$$+ 3k+1$$
$$= k^2 + 4$$

Consider

A Buss' reduction + Simple FPT Vertex Cover
(remove high-deg. vertices) branche over endpoints of edges

B same interleaved

A kernel $= (G_A, k)$ $\qquad$ 2^k search space $\rightsquigarrow$ No

in $\Theta(2^k(n+m))$

B kernel same. remove two edges $\rightsquigarrow$ parameter $k-2$

center of head has degree $k-\underline{1}$

also outer nodes in head have degree $k-\underline{1}$

↳ No, since edges in tail left.

Interleaving helps a lot here.

3.6 Lower Bounds by ETH

Definition 3.47 (Exponential-Time Hypothesis)
The *Exponential-Time Hypothesis (ETH)* asserts that there is a constant $\varepsilon > 0$ so that every algorithm for p-3SAT requires $\Omega(2^{\varepsilon k})$ time, where k is the number of variables.

$" c^k = (1+\delta)^k$

Alternative formulations:

- There is a $\delta > 0$ so that every 3-SAT algorithm needs $\Omega((1+\delta)^k)$ time.
- There is no $2^{o(k)}$-time algorithm for 3-SAT.
- There is no subexponential-time algorithm for 3-SAT.

Idea: Show that solving X in time $f(k, n)$ implies a $\mathcal{O}(2^{\varepsilon k} n^c)$ algorithm for 3SAT *for all* $\varepsilon > 0$.
$\rightsquigarrow$ unless ETH fails, no such $f(k, n)$-time algorithm for X exists. $\Rightarrow$ ETH $\lightning$

Problem: Need a reduction that preserves parameter k.

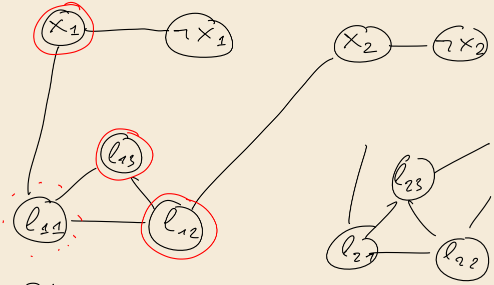
Recap: Reduction from 3SAT to Vertex Cover

$\varphi = \{\{\ell_{11}, \ell_{12}, \ell_{13}\}, \{\ell_{21}, \ell_{22}, \ell_{23}\}, \ldots\}$

Gadgets

Variables

Clauses



φ n clauses

k variables

$|V| = 3n + 2k$

Threshold $\quad 2n + k = k' = \Theta(n)$

(G, k') Yes iff φ satisfiable

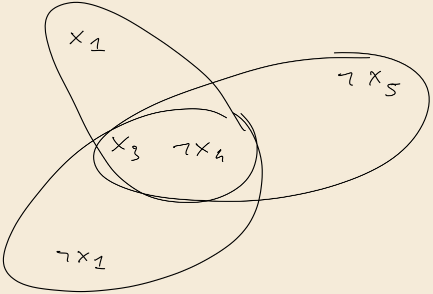
Lemma 3.48 (Sparsification Lemma)

For all $\varepsilon > 0$, there is a constant K so that we can compute for every formula φ in 3-CNF with n clauses over k variables an equivalent formula $\bigvee_{i=1}^{t} \psi_i$ where each ψ_i is in 3-CNF and over the same $\cancel{n}$ k variables and has $\leq K\cancel{n}$ k clauses. Moreover, $t \leq 2^{\varepsilon k}$ and the computation takes $\mathcal{O}(2^{\varepsilon k} n^c)$ time. ◀

Rough Idea:

Iteratively remove *sunflowers* by retaining only the *heart* or only the *petals*.

$$\varphi = \{\{x_1, x_3, \neg x_4\}, \{\neg x_5, x_3, \neg x_4\}, \{\neg x_1, x_3, \neg x_4\}\}$$



Theorem 3.49 (Lower Bound by Size)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-3SAT needs time $\Omega(2^{c(n+k)})$ where n is the number of clauses and k is the number of variables. ◀

Proof: Assume $\forall c > 0$ then A_c that solves 3SAT in
$O(2^{c(n+k)} n^b)$.

Let $\delta > 0$ given. to show: B_δ that solves 3SAT
$O(2^{\delta k} \cdot n^b)$ $\lightning$ ETH

B_δ $\varepsilon = \frac{\delta}{2}$ $\rightsquigarrow$ K from sparsification lemma

(1) $\varphi \rightsquigarrow \bigvee \psi_i$ with $\varepsilon = \frac{\delta}{2}$

$|\psi_i| \leqslant Kk$

(2) For each ψ_i A_c with $c = \frac{\delta}{2(K+1)}$

(3) Return Yes iff $\exists i$ ψ_i satisfiable

Time: (1) $O(2^{\varepsilon k} n^b) = O(2^{\frac{\delta}{2}k} n^b)$

(2) $2^{\varepsilon k} \cdot O(2^{c(|\psi|+k)} n^b) = O\left(2^{\frac{\delta}{2} \cdot k + \frac{\delta}{2\cancel{(k+1)}} \cdot (\cancel{(k+1)} k)} n^b\right)$

$= O(2^{\delta k} n^b)$

total $O(2^{\delta k} n^b)$

$\square$

Theorem 3.50 (No Subexponential Algorithm Vertex Cover)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-VERTEX-COVER needs time $\Omega(2^{ck})$.

Proof: Assume A_c solves Vertex Cover in $O(2^{ck} n^b)$ time $\forall c > 0$

$\delta > 0$ given

B_δ (1) Construct (G, k) from φ (poly-time)

(2) Use A_c $\quad c = \frac{\delta}{B}$

$k = O(n)$

say $k \leq Bn$

B_δ solves 3-SAT

Time: $O(n^b) + O(2^{ck} n^b) = O(2^{\frac{\delta}{B} Bn} n^b) = O(2^{\delta n} n^b)$

$\Rightarrow$ "subexponential" algo (B_δ) for 3-SAT $\lightning$ Thm 3.49 $\wedge$ ETH $\square$

Theorem 3.51 (Lower Bound Closest String)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-Closest-String needs time $\Omega(2^{c(k \operatorname{ld} k)}) = \Omega(k^{ck})$.

4 Randomized Algorithms and Data Structures

Computational Models

We consider deterministic TM,
with an additional input tape (read-only, unidirectional)
containing an infinite sequence of random 0s and 1s.

$\rightsquigarrow$ Algorithms can use random bits to decide on branching.

$\Rightarrow$ Randomized algorithm A defines a random experiment on input x

probability space $(S_{A,x}, Prob)$

$S_{A,x} = \{ c \mid c$ randomly controlled computation of A on $x \}$

$Prob$ = probability distribution on $S_{A,x}$ induced by $0/1$ sequences

a) Complexity Measure

$$\text{Random}_A(n) = \max\{\text{Random}_A(x) \mid x \text{ with } |x|=n\}$$

$$\text{Random}_A(x) = \max\{\text{random bits used by } c \;;\; c \text{ computation of } A \text{ on } x\}$$

$\mathrm{Prob}_{A,x}(c)$ = prob. of sequence of random bits read by A on input x during computation c.

$$\mathrm{Prob}(A(x)=y) = \sum_{\substack{c \\ c \text{ outputs } y}} \mathrm{Prob}_{A,x}(c)$$

// probability of output x

length of computation c

$$\mathbb{E}\text{-}\mathrm{Time}_A(x) = \sum_c \mathrm{Prob}_{A,x}(c) \cdot \mathrm{Time}(c)$$

$$\mathbb{E}\text{-}\mathrm{Time}_A(n) = \max\{\mathbb{E}\text{-}\mathrm{Time}_A(x)\}$$

while (rbit() == 1) { }

usually well-defined even if infinite runs are possible

but hard to determine

Worst-Case Time

$$\mathrm{time}_A(x) = \max\{\mathrm{Time}(c) : c \text{ computation by } A \text{ on } x\}$$

$$= \infty \text{ if we have the possibility of infinite runs}$$

$$\mathrm{time}_A(n) = \max\{\mathrm{time}_A(x) : |x| = n\}$$