CS 341: Algorithms

Lecture 23: Misc

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

EXP and beyond

Exponential time

Definition

EXP is the set of decision problems that can be solved in exponential time $2^{O(\mathsf{size}(x)^k)}$ for some k.

Observation: **NP** $\subset$ **EXP** so problems in **NP** cannot be extraordinarily bad

Idea: brute-force, try all possible certificates

- for a given x, we look for a certificate of size $\mathsf{size}(x)^k$, for some constant k
- if we work with binary symbols, there are $2^{\mathsf{size}(x)^k}$ certificates
- each of them takes polynomial time

Bounded halting

Definition

- **instance:** program / Turing maching P, input x to P, integer t
- **output:** does $P(x)$ stop on input x within t steps?
- **remark:** input size $= \mathsf{size}(P) + \mathsf{size}(x) + \log t$

Claim

BoundedHalting is in **EXP**

Proof (sketch)

- use a **universal Turing machine**, run the simulation for t steps
- runtime polynomial can be made polynomial in $\mathsf{size}(P)$, t
- which is exponential in the input size

EXP-completeness

Claim

BoundedHalting is **EXP**-complete

Proof

- take decision problem Prob in **EXP**
- so there is a program / Turing machine P that decides Prob(x) using at most $2^{c\,\mathsf{size}(x)^k}$ operations (c, k constants)
- modify P to make it run forever if input **no**-instance, call P' the result
- **reduction:** on instance x, call BoundedHalting with input P', x and $t = 2^{c\,\mathsf{size}(x)^k}$
- P' is fixed, x is x and $\log t = c\,\mathsf{size}(x)^k$, so this is polynomial in $\mathsf{size}(x)$

Remark: because **NP** $\subset$ **EXP**, this shows BoundedHalting is **NP**-hard

Time hierarchy

Time hierarchy theorem (particular case)

$\mathbf{P} \neq \mathbf{EXP}$

Proof: take CS360

Consequence

$\textsc{BoundedHalting}$ is not in $\mathbf{P}$

Proof:

- if it was, using $\mathbf{EXP} \leq_P \textsc{BoundedHalting}$, we would get $\mathbf{P}{=}\mathbf{EXP}$

Even worse

HALTING

- **instance:** program / Turing maching P, input x to P
- **output:** does $P(x)$ stop on input x?
- **remark:** input size $=$ size(P) $+$ size(x)

1. **Undecidable** (CS245, CS360), so in particular not in **NP**

2. **RE-hard** (so in particular NP-hard, and **not** NP-complete)
   - take a **recursively enumerable** problem PROB
   - **meaning:** there is a program / Turing machine P st PROB(x) returns **true** for **yes**-instances, and either loops or returns **false** for **no**-instances
   - modify P to make it run forever if input **no**-instance, call P' the result
   - **reduction:** on instance x, call HALTING with input P', x
   - P' is fixed, x is x, so this is polynomial in size(x)

Variants of kSAT

Definitions

kSAT

- instance: a **boolean formula** in n variables $x_1, \ldots, x_n$ in **CNF**

$$(y_{\mathbf{1,1}} \vee \cdots \vee y_{\mathbf{1},\boldsymbol{k_1}}) \quad \wedge \quad \cdots \quad \wedge \quad (y_{\boldsymbol{\ell},\mathbf{1}} \vee \cdots \vee y_{\boldsymbol{\ell},\boldsymbol{k_\ell}})$$

  with literals $y_{i,j}$ of the form x_m, $\overline{x_m}$ and $k_i \leq k$

- **problem:** is there a choice of the variables that makes it true?

EXACT-kSAT

- same as above, but with **exactly** k literals (repetitions OK)

UNIQUE-kSAT

- same as above, but with **exactly** k literals and **no repeated variable**
- for $\boldsymbol{k} = \mathbf{3}$, $x \vee y \vee \overline{z}$ OK, $x \vee x \vee z$ not OK, $x \vee \overline{x} \vee z$ not OK

Equivalence

Claim

$\textsc{exact-k}\text{SAT} \leq_P \textsc{k}\text{SAT} \leq_P \textsc{exact-k}\text{SAT}$

Proof:

1. an $\textsc{exact-k}\text{SAT}$ instance is a $\textsc{k}\text{SAT}$ instance
2. transform $x \vee y$ into $x \vee x \vee y$

Claim

$\textsc{unique-k}\text{SAT} \leq_P \textsc{k}\text{SAT} \leq_P \textsc{unique-k}\text{SAT}$

Proof:

1. a $\textsc{unique-k}\text{SAT}$ instance is a $\textsc{k}\text{SAT}$ instance
2. transform $x \vee y$ into $(x \vee y \vee \text{dummy}) \wedge (x \vee y \vee \overline{\text{dummy}})$

2SAT and MAX-2SAT

2SAT is in P

Remark: any $\textsc{k}$SAT is in **NP**

- **instance:** formula F in kCNF
- **certificate $\boldsymbol{y}$:** boolean values for the variables that appear in F
- **algorithm $\boldsymbol{B(F, y)}$:** test if $F(y)$ is **true** (i.e. if all clauses are **true**)
- **NP**? yes! B runs in polynomial time, and F is satisfiable iff there exists a certificate of size $\leq \text{size}(F)$

We know: 3SAT NP-complete (and so $\textsc{k}$SAT as well, for $k \geq 3$)

Claim:

2SAT in **P**

Proof: we start from a formula F in 2CNF that has $\boldsymbol{s}$ clauses

assume all clauses have 2 literals

Introducing a graph

Idea: $x_i \vee x_j$ is equivalent to

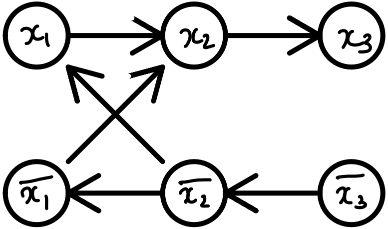
$$\overline{x_i} \implies x_j \quad \text{and to} \quad \overline{x_j} \implies x_i$$

- we can chain these implications to eventually find out a satisfiable solution
- so we put them in a directed graph G (with vertices labeled x_i and $\overline{x_i}$)

Example

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

gives



How to use the graph

Observation: suppose booleans $y_1, \ldots, y_n$ satisfy F

- assigns boolean values to all vertices
- if vertex v is true and $v \to w$ edge, w true — because $\overline{v} \vee w$ clause in F
- so if v is true and $v \rightsquigarrow w$ path, w true

Consequence: if some $x_i, \overline{x_i}$ are in the same SCC of G, F not satisfiable

Decision algorithm:

- construct G (at most $2s$ vertices and $2s$ edges)
- find the SCCs of G (= put indices on vertices)
- if any $x_i, \overline{x_i}$ that appear in F have the same index, return **false**
- else, return **true**

Runtime: $\boldsymbol{O(s)}$ in the word RAM model, **polynomial** in $\boldsymbol{s \log n}$ in the bit model

Proof + finding satisfying assignments

Algorithm, cont. (assuming **true**)

- **contract** the SCCs of G to obtain a DAG G'
- find a topological order o on G'
- for $i = 1, \ldots, n$
  - if $o(x_i) < o(\overline{x_i})$, take $y_i =$ **false**
  - if $o(\overline{x_i}) < o(x_i)$, take $y_i =$ **true**
  - if $o(x_i)$ undefined, y_i arbitrary

(still polynomial time)

Claim: $F(y_1, \ldots, y_n) =$ **true**

Proof: suppose that $x_i \vee x_j$ clause not satisfied, so x_i and x_j assigned **false**

- so $o(x_i) < o(\overline{x_i})$ and $o(x_j) < o(\overline{x_j})$
- $(\overline{x_i}, x_j)$ edge, so $o(\overline{x_i}) \leq o(x_j)$ and $\boldsymbol{o(\overline{x_i}) < o(\overline{x_j})}$
- $(\overline{x_j}, x_i)$ edge, so $o(\overline{x_j}) \leq o(x_i)$ and $\boldsymbol{o(\overline{x_j}) < o(\overline{x_i})}$

contradiction

MAX-kSAT

k-terms conjonctive formula satisfiability, optimization version:

- **instance:** a boolean formula $\boldsymbol{F}$ in n variables $x_1, \dots, x_n$ in kCNF
- **problem:** find the maximal number of clauses that can be satisfied simultaneously

Decision version: MAX-κSAT

- **instance:** F as above, and an integer K
- **problem:** is there a choice of the variables that satisfies at least K clauses?
- **certificate:** boolean values for the variables that appear in F
- **algorithm $\boldsymbol{B}$:** count if at least K clauses in $F(y)$ are true

We prove: MAX-2SAT NP-complete

Exercise

we already could tell that MAX-κSAT NP-complete for $k \geq 3$

3SAT $\leq_P$ MAX-2SAT

Preliminaries:

- consider a clause $C = x \vee y \vee z$ (repeated variables OK)
- introduce a new variable t, and the 10 clauses

$$x,\ y,\ z,\ t,\ \overline{x} \vee \overline{y},\ \overline{y} \vee \overline{z},\ \overline{z} \vee \overline{x},\ x \vee \overline{t},\ y \vee \overline{t},\ z \vee \overline{t}$$

Claim

- you cannot satisfy more than 7 of these new clauses
- a boolean assignment of x, y, z, t that satisfies 7 clauses makes C **true**
- given a boolean assignment for x, y, z that makes C **true**, you can find a value for t that satisfies 7 clauses

case discussion (discuss whether $0, 1, 2$ or 3 of x, y, z are **true**)

3SAT $\leq_P$ MAX-2SAT

Reduction. Given a family F of k clauses that form a 3SAT problem, introduce

- one new variable t_i per clause in F,
- the 10 clauses as seen before (per clause in F)
- $K = 7k$

(takes polynomial time)

Correctness:

- you cannot satisfy more than $7k$ of these new clauses
- you satisfy $7k$ of them simultaneously if and only if you can satisfy all k input clauses simultaneously

Conclusion: MAX-2SAT is NP-complete

Randomization and approximation

Using randomization (for the optimization problem)

MAX-UNIQUE-3SAT

- **input:** F in 3CNF, with 3 distinct variables per clause (works for any k)
- **problem:** find the maximal number of clauses that can be satisfied simultaneously
- decision version NP-complete

Claim

using in expected polynomial time in n, s, we can find an assignment that satisfies at least 87.5% of the clauses

RandomAssignment(F)
1. F formula in 3CNF, 3 distinct variables per clause, $\boldsymbol{s}$ clauses
2. **repeat**
3. pick $x_1, \ldots, x_n$ uniformly at random in $\{0, 1\}$
4. **until** at least $7s/8$ clauses are satisfied
5. **return** $x_1, \ldots, x_n$

Analysing a single assignment

Definition: for $i = 1, \ldots, s$, let X_i be the indicator random variable

- $X_i = 0$ if ith clause is not satisfied
- $X_i = 1$ if ith clause is satisfied

Analysis:

- clause i has 3 variables and out of the 8 possibilities, only 1 makes it **false**
- so $p(X_i = 1) = 7/8$
- so $E[X_i] = 7/8$

Looking at all clauses:

- the number N of satisfied clauses is $\sum_{i \leq s} X_i$
- so $E[N] = 7s/8$

Overall runtime

Defining p

- let $\boldsymbol{p}$ be the probability that a random assignment satisfies at least $7s/8$ clauses
- then the **expected number of attempts** is

$$p + 2p(1-p) + 3p(1-p)^2 + \cdots = \frac{\mathbf{1}}{\boldsymbol{p}}$$

- and the expected runtime is $\boldsymbol{O((n+s)/p)}$ (in the word RAM model)

Introducing $p_0, \ldots, p_s$ and s'

- for $j = 0, \ldots, s$, let $\boldsymbol{p_j}$ be the probability that we satisfy $\boldsymbol{j}$ clauses
- let $\boldsymbol{s'}$ be the largest integer **less than** $7s/8$

Consequences

- $\boldsymbol{s' \leq 7s/8 - 1/8}$
- $\boldsymbol{p = \sum_{j \geq s'+1} p_j}$

Overall runtime

$$
\begin{aligned}
\frac{7}{8}s &= E[N] \\
&= \sum_j j p_j \\
&= \sum_{j \le s'} j p_j + \sum_{j \ge s'+1} j p_j \\
&\le \sum_{j \le s'} s' p_j + \sum_{j \ge s'+1} s p_j && j \le s',\ j \le s \\
&= s'(1-p) + sp && \text{previous slide} \\
&\le s' + sp && 1 - p \le 1 \\
&\le \frac{7}{8}s - \frac{1}{8} + sp && \text{previous slide}
\end{aligned}
$$

Finally: $1/8 \le sp$ so $\boldsymbol{1/p \le 8s}$

Bonus

Medium: derandomize the algorithm

- assign one variable at a time
- at the beginning,

$$\frac{7}{8}s = E[N] = \frac{1}{2}E[N|x_1 = 0] + \frac{1}{2}E[N|x_1 = 1]$$

  so one of $E[N|x_1 = 0]$ and $E[N|x_1 = 1]$ must be at least $\frac{7}{8}s$
- both can be computed in polynomial time, choose the better one and continue

Extra hard: beat 7/8

- if there is a polynomial-time algorithm that finds a fraction $7/8 + \varepsilon$ of the optimal, then **P=NP**