Algorithm Theory

Georg Moser

Institute of Computer Science @ UIBK

Summer 2007

Content

Pseudopolynomial Algorithms

given an instance of KNAPSACK with

1. n items
2. values: $\{v_1, \dots, v_n\}$
3. weights: $\{w_1, \dots, w_n\}$

we seek $S \subseteq \{1, \dots, n\}$ such that $\sum_{j \in S} v_j \geqslant K$

Algorithm:

1. $V = \max\{v_1, \dots, v_n\}$ and $W = \max\{w_1, \dots, w_n\}$
2. set $V(w, 0) = 0$ for all w
3. $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

note that

$$V(w, i) = \max(\{\textstyle\sum_{j \in S} v_j \mid S \subseteq \{1, \dots, i\} \text{ and } \sum_{j \in S} w_j = w\}$$

4. to solve KNAPSACK is suffices to pick an entry greater than or equal the goal K; this can be done in time $\mathcal{O}(nW)$

Strong **NP**-completeness

Observation

- all the **NP**-completeness problems considered (except KNAPSACK) used polynomially small integers (in the size of the input)
- the **NP**-completeness proof for KNAPSACK needed exponentially large integers

Definition — strongly **NP**-complete

a problem is called strongly **NP**-complete if

- any instance x with $n = |x|$ contains integers of size at most $p(n)$ for a polynomial p

Theorem

CIRCUIT SAT, SAT, ..., INDEPENDENT SET, ..., EXACT COVER BY 3-SETS, ... are all strongly **NP**-complete

NP and coNP

Definition (polynomial verifier)

- A verifier of a language L is an algorithm P such that:

$$L = \{w \mid \text{there exists a string } c \text{ so that P accepts } \langle w, c\rangle\}$$

- A polynomial verifier is one that runs in time polynomial in $|w|$

Definition (succinct certificates)

L has succinct certificates (the string c)
if $\exists$ polynomial verifier for L
(recall that $|c| \leqslant p(|w|)$ for some polynomial p)

Definition (coNP)

$$\mathbf{coNP} = \{\overline{L} \colon L \in \mathbf{NP}\}$$

Observation

- suppose $L \in \mathbf{coNP}$, and a string x such that $x \notin L$
- then $x \in \overline{L}$, which implies the existence of a succinct certificate c
- hence the "no"-instance x has a succinct disqualification

Example

$$\text{VALIDITY} = \{\varphi \colon \varphi \text{ is a valid CNF-formula}\}$$

1. if φ is not valid, then the disqualification is an assignment T, such that $T(\varphi) = \textbf{false}$
2. the disqualification is succinct, i.e. it is at most polynomial in the length of the formula

Example

HAMILTON PATH COMPLEMENT =
$\{G \colon G$ is a directed graph without Hamilton path$\}$

Theorem

if L is **NP**-complete, then its complement $\overline{L}$ is **coNP**-complete.

Example

VALIDITY and HAMILTON PATH COMPLEMENT are examples of **coNP**-complete problems

Proof Sketch

we indicate the pattern of the proof for VALIDITY

1. we show the existence of a log-space reduction R such that for every $L \in \mathbf{coNP}$: $x \in L$ iff $R(x) \in$ VALIDITY:

$x \in L$	iff	$x \notin \overline{L}$	Note that $\overline{L} \in \mathbf{NP}$
	iff	$S(x) \notin$ SAT	**NP**-completeness of SAT
	iff	$\neg S(x)$ is valid	S a log-space reduction
	iff	$\neg S(x) \in$ VALIDITY	

2. set $R(x) := \neg S(x)$ □

Theorem

if a **coNP**-complete problem L is in **NP**, then **NP** = **coNP**

Proof

we show: $\mathbf{coNP} \subseteq \mathbf{NP}$:

- consider $L' \in \mathbf{coNP}$
- $\exists$ reduction R from L' to L □

Theorem

if $\exists$ **NP**-complete problem L such that its complement $\overline{L}$ is in **NP**, then **NP** = **coNP**

Theorem

if $\mathbf{NP} \neq \mathbf{coNP}$, then $\mathbf{P} \neq \mathbf{NP}$

Proof

1. assume to the contrary that $\mathbf{P} = \mathbf{NP}$
2. as **P** is closed under complement, we have $\mathbf{P} = \mathbf{coP} = \mathbf{coNP}$
3. hence, we conclude $\mathbf{NP} = \mathbf{coNP}$ □

Observation — NP ∩ coNP

- problems in **NP** have succinct certificates
- problems in **coNP** have succinct disqualifications
- thus for $L \in \mathbf{NP} \cap \mathbf{coNP}$ each *yes* instance as a succinct certificate and each *no* instance has a succinct disqualification clearly no instance has both

Example

consider the language PRIMES:

$$\text{PRIMES} = \{p \colon p \text{ is a prime number}\}$$

Theorem — Pratt

PRIMES $\in$ **NP** $\cap$ **coNP**, i.e., for each number n:

- either, we have a certificate that shows that n is not a prime
- or, we have a certificate that shows that n is a prime

Disqualification & Qualification for PRIMES

Fact

the obvious $\mathcal{O}(\sqrt{n})$ is pseudopolynomial
$\sqrt{n}$ is not a polynomial in $|(n)_2|$

Theorem

PRIMES $\in$ **coNP**

Proof

given p

- the string that disqualifies p is a pair $((u)_2, (v)_2)$ such that $p = u \cdot v$
- the length of the disqualification is polynomial in the length of p

□

Theorem

A number $p > 1$ is prime iff there is a number $r \in \{2, \dots, p-1\}$ such that $r^{p-1} = 1 \mod p$, and $r^{\frac{p-1}{q}} \neq 1 \mod p$ for all prime divisors q of $p-1$; r is called primitive root

Proof Idea

employ Fermat's (small) Theorem
for all $r \in \{1, \dots, p-1\}$: $r^{p-1} = 1 \mod p$ □

Theorem

PRIMES $\in$ **NP**

Proof

the certificate consists of

1. the primitive root r
2. the prime divisors $q_1, \dots, q_k$
3. primality certificates for q_i

Nondeterministic Algorithm

given p (in binary)

1. if $p = 2$, accept; if $p > 2$ and p even, reject
2. guess prime factorisation of $p - 1 = {q_1}^{k_1} \cdots {q_m}^{k_m}$ verify by multiplication
3. guess $r \in \{2, \dots, p-1\}$ and verify that $r^{p-1} = 1 \mod p$
4. verify for each i: $r^{\frac{p-1}{q_i}} \neq 1 \mod p$
5. recursively verify that $q_1, \dots, q_m$ are prime

Observation

step 1 is constant; step 2 polynomial in $\log p$; steps 3 & 4 can be performed in polytime (in $\log n$) by repeatedly squaring; solving the recursion yields a polytime algorithm

Succinct Certificate

alternatively the data of the algorithm can be collected in a succint certificate $C(p) = (r; q_1, C(q_1), \dots)$

$$C(67) = (2; 2, (1), 3, (2; 2, (1)), 11, (8; 2, (1), 5, (3; 2, (1))))$$

□

Theorem

Agrawal, Kayal, Saxena

PRIMES $\in$ **P**

“Proof Idea

suppose a and p are coprime, then p is prime iff

$$(x - a)^p \equiv (x^p - a) \mod p$$

□