Chap 34 – NP-Completeness

34.1 Polynomial time

34.2 Polynomial-time verification

34.3 NP-completeness and reducibility

34.4 NP-completeness proofs

34.5 NP-completeness problems

Decision problems

- Decision problem

  The theory of NP-Completeness focuses on the complexities of decision problems, e.g.

  - The prime problem (PRIME)

    Is $n \geq 2$ a prime?

  - The searching problem (SEARCH)

    Does $y \in \{x_1, x_2, \ldots, x_n\}$?

  - The sorting problem (SORT)

    Is $(y_1, y_2, \ldots, y_n)$ a sorted permutation of $(x_1, x_2, \ldots, x_n)$?

  - The tautology problem (TAUT)

    Is the formula F in propositional logic a tautology?

Decision problems

- Decision problem and optimization problem
  - Each optimization problem has a related decision problem.
  - By showing that a decision problem is hard, we know that the corresponding optimization problem is hard, too.
  - Example

    The knapsack optimization problem

    Maximize $\sum_{i=1}^{n} v_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leq W$

    where $x_i = 0$ or 1

Decision problems

- Decision problem and optimization problem
  - Example (Cont'd)

    The knapsack decision problem (KNAPSACK)

    Given W, w_i, v_i and a **lower bound** B, do there exist $x_i = 0$ or $1, 1 \leq i \leq n$, such that

    $$\sum_{i=1}^{n} w_i x_i \leq W \quad \text{and} \quad \sum_{i=1}^{n} v_i x_i \geq B$$

    Comment

    To obtain the related decision problem:

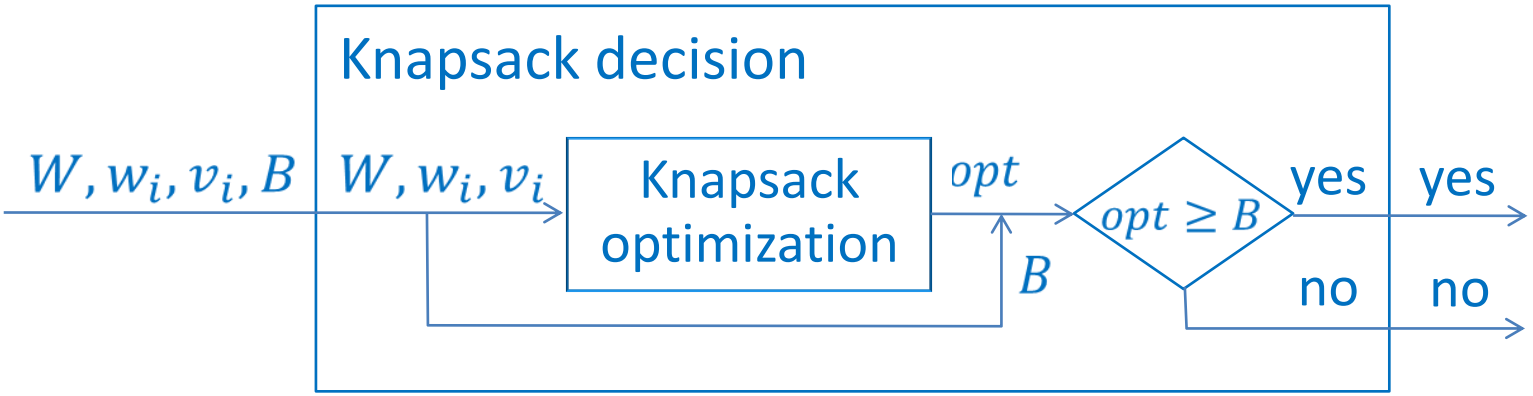
    - For maximization problem, add a lower bound
    - For minimization problem, add an upper bound

Decision problems

- Decision problem and optimization problem
  - Example (Cont'd)

    Reduction: "Reduce problem A to problem B" means "Use an algorithm for B to construct an algorithm for A".

    For example, we may reduce the decision problem to the optimization problem as follows:



Decision problems

- Decision problem and optimization problem
  - Example (Cont'd)

    This algorithm of solving the decision problem has the same time complexity as that of the optimization problem.

    However, there might have a better algorithm for solving the decision problem.

    Thus, a decision problem is ***no harder* (*perhaps, easier*) *than*** the related optimization problem, i.e.

    decision $\leq$ optimization

    Thus, if a decision problem is shown to be hard, the related optimization problem is hard, too.

34.1 Polynomial time

- Encoding
  - Example

    // Classic primality test

    PRIME(n)

    for $d = 2$ **to** $\lfloor \sqrt{n} \rfloor$ **do**

        if $\mathrm{mod}(n, d) == 0$ **then return** false

    return true

    Clearly, the worst-case running time is $\Theta(n^{0.5})$.

    But, this is NOT a polynomial-time algorithm.

    Indeed, it is an exponential time algorithm and runs very slow.

34.1 Polynomial time

- Encoding
  - Example (Cont'd)

    Suppose that n occupies 25 bytes.

    Let $n \approx 2^{200} \approx 10^{60}$ be a prime

    On a 1TIP (10^{12} instructions per second) machine

    running time $\geq \sqrt{10^{60}}/10^{12} = 10^{18}$ seconds

    $\geq 10^{18}/10^{5} = 10^{13}$ days

    $\geq 10^{13}/10^{3} = 10^{10}$ years

    Thus, to determine if PRIME takes an exponential time, we can't look at the function on n. Instead, we have to look at the function on the size of the storage occupied by n.

34.1 Polynomial time

- Encoding
  - Example (Cont'd)

    With binary encoding, the input size is the number of bits occupied by the input.

    Since the integer n occupies $\lg n$ bits, it follows that

    $$O(\sqrt{n}) = O\left(\sqrt{2^{\lg n}}\right)$$

    is an exponential-time algorithm.
  - Encoding

    e : Abstract problem $Q \rightarrow$ concrete problem $e(Q)$

    For example, the binary encoding e for the PRIME problem:

    $e : \{2,3,4,\ldots\} \rightarrow \{10_2, 11_2, 100_2, \ldots\}$

34.1 Polynomial time

- Encoding
  - Encoding (Cont'd)

    $Q : I = \{\text{abstract problem instances}\} \rightarrow \{0,1\}$

    $\downarrow$ encoding e

    $e(Q) : e(I) = \{\text{concrete problem instances}\} \rightarrow \{0,1\}$
  - The time complexity of an algorithm is measured by $|e(i)|, i \in I$.
  - **DEFINITION**

    A concrete problem is polynomial-time solvable if there exists an algorithm to solve it in time $O(n^k)$ for some k where $n = |e(i)|$

34.1 Polynomial time

- The complexity class P
  - $\mathrm{P} = \{e(Q) \mid e(Q) \text{ is polynomial-time solvable}\}$
  - Example – Assume binary encoding

    Whether PRIME $\in$ P has long been unknown.

    AKS primality test shows that PRIME $\in$ P
    - published on August 2002
    - runs in $\tilde{O}(\lg^{12} n) \rightarrow \tilde{O}(\lg^{6} n)$ polynomial time

      where the "Soft O" asymptotic notation $\tilde{O}$ means

      $\tilde{O}(f(n)) = O\left(f(n)\lg^{k} f(n)\right)$ for some k

34.1 Polynomial time

- The complexity class P
  - Example (Cont'd)

    Although AKS primality test runs in polynomial time, it is still impractical.

    Suppose that n occupies 25 bytes.

    Let $n \approx 2^{200} \approx 10^{60}$ be a prime

    On a 1TIP (10^{12} instructions per second) machine

    running time $\geq (\lg 2^{200})^6 / 10^{12}$

    $= 200^6 / 10^{12}$

    $= 64$ seconds

    In practice, still use probabilistic primality test.

34.1 Polynomial time

- The complexity class P
  - Example

    Whether KNAPSACK $\in$ P is still unknown.

    The dynamic programming solution runs in $O(nW)$ time.

    (Exercise 16.2-2)

    Due to the number W, $O(nW)$ is exponential. (Ex 34.1-4)

$$O(nW) \neq O\left(\left(\underbrace{2n+1}_{} + \lg B + \lg W + \sum_{k=1}^{n}(\lg v_i + \lg w_i)\right)^k\right)$$

    for any k.

    $\because$ need $2n+1$ separators to separate $2n+2$ numbers

    (See next example for more explanations.)

34.1 Polynomial time

- The complexity class P
  - Example (Cont'd)
    - $O(nW)$ is called pseudo-polynomial, i.e. it would be polynomial if an upper bound were imposed on W, e.g.

$$W = O(n^k)$$

$$\Rightarrow O(nW) = O(n^{k+1})$$

$$= O\left(\left(2n + 1 + \lg B + \lg W + \sum_{k=1}^{n} (\lg v_i + \lg w_i)\right)^{k+1}\right)$$

    - KNAPSACK and PRIME are called number problems; but SORT and SEARCH aren't.

34.1 Polynomial time

- The complexity class P
  - Example

    SEARCH ∈ P SEARCH ≡ Does $y \in \{x_1, x_2, \ldots, x_n\}$?

    The $O(n)$ sequential search algorithm is polynomial-time.

    $$\because O(n) = O\left(n + \lg y + \sum_{i=1}^{n} \lg x_i\right)$$

    Let $\{0,1,\#\}$ be the alphabet of the concrete instances

    An abstract instance $y \in \{x_1, x_2, \ldots, x_n\}$ may be encoded as

    $e(y)\#e(x_1)\#\cdots\#e(x_n)$

    where e is the binary encoding.

    Note that there are n #'s.

34.1 Polynomial time

- Polynomially-related encodings
  - Example (Cont'd)

    The preceding encoding has several interpretations.

    All of them don't affect the algorithm's complexity.
    - Ternary encoding with three symbols 0, 1, and #

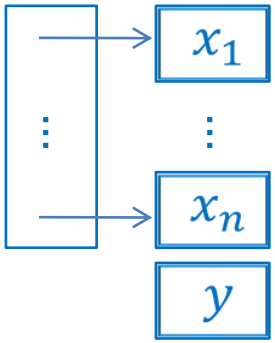
      (Numbers are encoded in binary).
    - Binary encoding: 0, 1, and # are characters.

      Using 7-bit ASCII code, the size of a concrete instance

$$= 7\left(n + \lg y + \sum_{i=1}^{n} \lg x_i\right)$$

34.1 Polynomial time

- Polynomially-related encodings
  - Example (Cont'd)
    - Binary encoding:



Using 32-bit pointers, the size of a concrete instance

$$= 32n + \lg y + \sum_{i=1}^{n} \lg x_i$$

34.1 Polynomial time

- Polynomially-related encodings
  - However, unary encoding affects the complexity.

    Let n be represented by 111 … 1 (i.e. n 1's)

    Then, for classic primality test, $O(\sqrt{n})$ is polynomial-time.

    We shall rule out unary encoding, since it is too expensive.
  - **DEFINITION** (Polynomially-related encodings)

    Two encodings e_1 and e_2 are polynomially related if there exist a polynomial-time computable function

    $f: e_1(Q) \to e_2(Q)$ and $g: e_2(Q) \to e_1(Q)$

    such that

    $f(e_1(i)) = e_2(i)$ and $g(e_2(i)) = e_1(i)$ for all $i \in I$

34.1 Polynomial time

- Polynomially-related encodings
  - Example
    - Binary and ternary encodings are polynomially related. Given a binary string of size n, the corresponding ternary string is of size $O(\log_3 2 \cdot n)$, which can be computed in polynomial time.
      $[N.B. 3^m = 2^n \Rightarrow m = \log_3 2 \cdot n]$
    - Binary and unary encodings aren't polynomially related.
    - Given a binary string of size n, the corresponding unary string is of size $O(2^n)$, which can't be computed in polynomial time.

34.1 Polynomial time

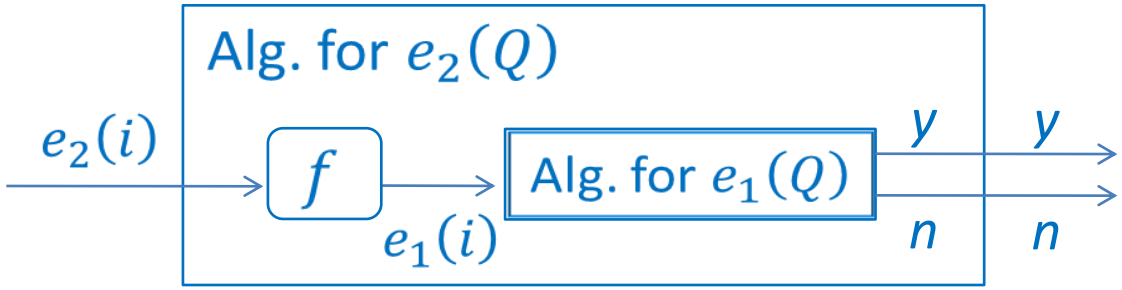
- Polynomially-related encodings
  - **LEMMA**

    Let e_1 and e_2 be polynomially related encodings. Then,

    $e_1(Q) \in P \Leftrightarrow e_2(Q) \in P$

    Proof

    We show that $e_1(Q) \in P \Rightarrow e_2(Q) \in P$ ($\Leftarrow$ is similar)



    Suppose f takes a time in $O(n^c)$, then

    $|e_1(i)| = O(|e_2(i)|^c)$

34.1 Polynomial time

- Polynomially-related encodings
  - **LEMMA** (Cont'd)

    Assume that Alg. for $e_1(Q)$ takes a time in $O(n^k)$.

    Then, for the instance $e_1(i)$, it takes a time in

    $O(|e_1(i)|^k) = O(|e_2(i)|^{ck})$

    Therefore,

    total time taken by Alg. for $e_2(Q)$

    $= O(|e_2(i)|^c) + O(|e_2(i)|^{ck})$

    $= O(|e_2(i)|^{\max\{c,ck\}})$

34.1 Polynomial time

- Polynomially-related encodings
  - Standard encoding $\langle i \rangle, i \in I$

    The standard encoding is understood to be the binary encoding as illustrated in p.14.
  - As long as we use an encoding e that is polynomially related to the standard encoding, we may define:

    An abstract problem Q is polynomial-time solvable if $e(Q)$ is polynomial-time solvable.
  - $\mathrm{P} = \{Q \mid Q \text{ is polynomial-time solvable}\}$

34.1 Polynomial time

- Formal language framework
  - Let $\Sigma = \{0,1\}$ be the alphabet of the standard encoding

    A decision problem Q may be viewed as a language over Σ.

    $L(Q) = \{\langle i \rangle \in \Sigma^* \mid Q(i) = 1\}$

    $=$ the language of all yes instances
  - Example

    The PRIME problem has the corresponding language

    $\text{PRIME} = \{\langle i \rangle \in \Sigma^* \mid i \text{ is a prime}\}$

    $= \{10,11,101,111,1011, \dots\}$
  - Solving a decision problem by an algorithm

    $=$ Accepting a language by the algorithm

34.1 Polynomial time

- Formal language framework
  - **DEFINITION**

    An algorithm A ***accepts*** a language

    $L(A) = \{x \in \Sigma^* \mid A \text{ terminates with } A(x) = 1\}$

    An algorithm A ***decides*** a language L if $L = L(A)$ and

    $\forall x \notin L, A$ terminates with $A(x) = 0$
  - Example

    PRIME(n) ◁ This algorithm decides the language PRIME.

    for $d = 2$ **to** $\lfloor \sqrt{n} \rfloor$

    if $\text{mod}(n, d) == 0$ **then return** false; // **while** (true);

    return true

    If replaced, accept PRIME

34.1 Polynomial time

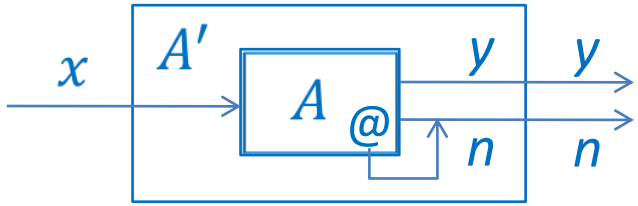
- Formal language framework
  - $\mathrm{P} = \{L \mid L \text{ is decided by a polynomial-time algorithm}\}$
  - **THEOREM**

    $\mathrm{P} = \{L \mid L \text{ is accepted by a polynomial-time algorithm}\}$

    decided $\subseteq$ accepted, trivial

    accepted $\subseteq$ decided

    Suppose that algorithm A accepts language L with at most cn^k time, where $n = |x|, x \in L$



After cn^k time, force A to terminate with answer no.

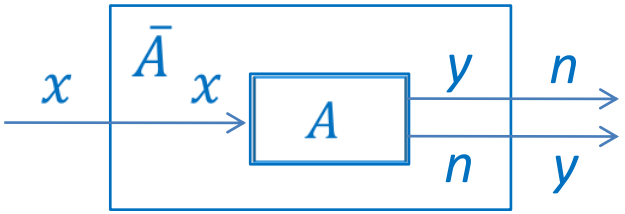
Thus, algorithm A' decides L in polynomial time.

34.1 Polynomial time

- The complexity class co-P
  - co-P $= \{\bar{L} \mid L \in \mathrm{P}\}$
  - **THEOREM** $\mathrm{P} = \text{co-P}$

    We show that $L \in \mathrm{P} \Leftrightarrow \bar{L} \in \mathrm{P}$ so that $\bar{\bar{L}} = L \in \text{co-P}$

    $\Rightarrow$ Suppose that algorithm A decides L in polynomial time.



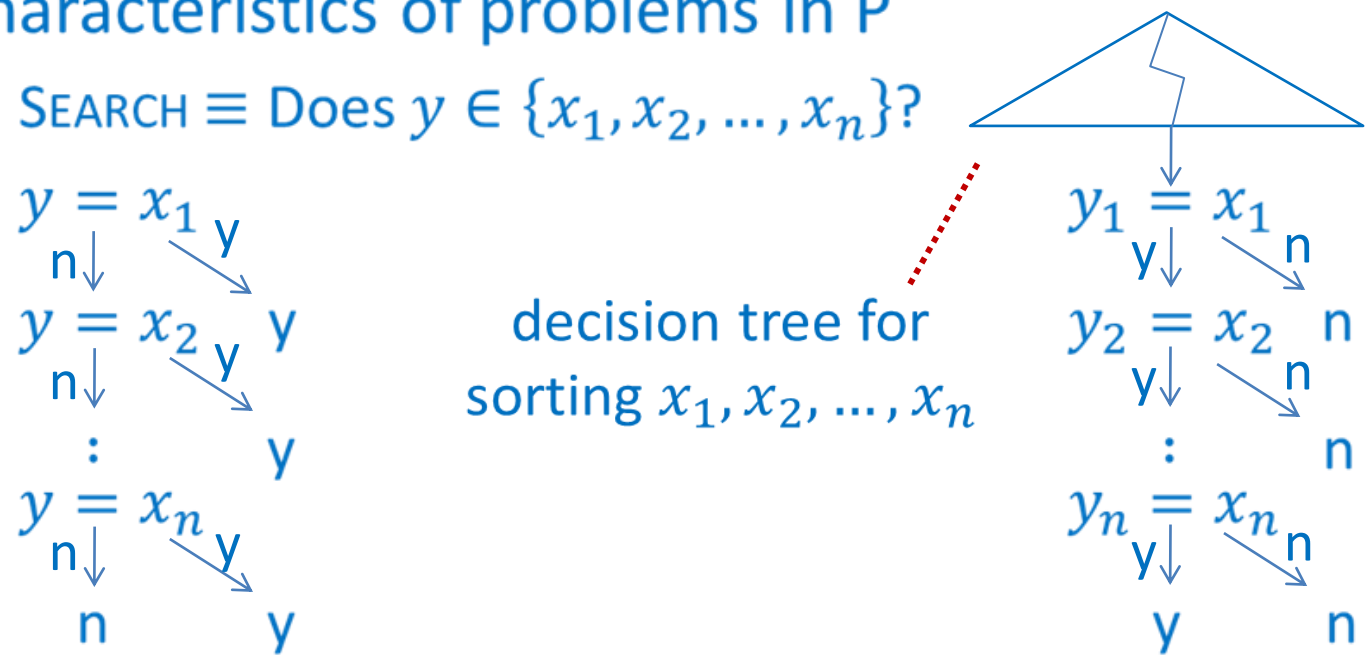
$$\bar{A}(x) = 1 \Leftrightarrow A(x) = 0$$

    Thus, algorithm $\bar{A}$ decides $\bar{L}$ in polynomial time.

    $\Leftarrow$ Similar
  - Ex: PRIME $\in$ P $\Rightarrow$ COMPOSITE $\in$ co-P $\Rightarrow$ COMPOSITE $\in$ P

34.2 Polynomial-time verification

- Characteristics of problems in P
  - SEARCH ≡ Does $y \in \{x_1, x_2, \ldots, x_n\}$?

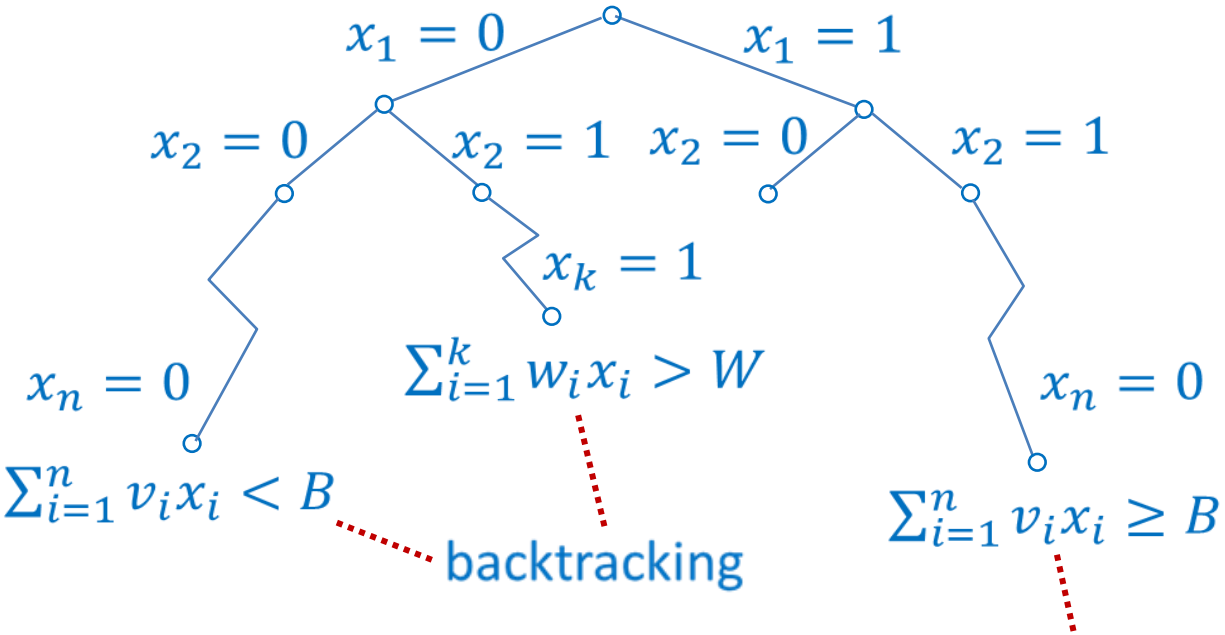


SORT ≡ Is $(y_1, \ldots, y_n)$ a sorted permutation of $(x_1, \ldots, x_n)$?

For each problem, there is only one search path for an instance. Moreover, the search path can be searched in polynomial time.

34.2 Polynomial-time verification

- Characteristics of problems possibly not in P
  - KNAPSACK – Given W, w_i, v_i and a **lower bound** B



There are 2^n paths to search.

However, each path can be searched in polynomial time

34.2 Polynomial-time verification

- Deterministic algorithm
  - Search the tree with backtracking
  - Complexity

$$\text{time spent} = \sum_{all\ paths} \text{time spent on a path}$$

$$= \text{exponential time in the worst case}$$

  - For the preceding backtracking algorithm for KNAPSACK, the worst-case running time is $O(n2^n)$.

34.2 Polynomial-time verification

- Nondeterministic algorithm
  - Search the tree ***nondeterministically*** at each decision point so that there are many searching processes – some succeeds and some fails.
  - Nondeterministic algorithms are theoretical.
    - They are meant to capture the features that there are many searching processes.
    - There is no concern of how to perform these searching processes, e.g. by parallel or deterministic left-to-right search, etc.

34.2 Polynomial-time verification

- Nondeterministic algorithm
  - For a yes-instance

    time spent

    = time spent on a shortest successful path

    = polynomial time in the worst case
  - For a no-instance

    time spent = undefined, because
    - to be consistent with a yes-instance, the time spent shall be defined as the time spent on some unsuccessful path, but
    - a single unsuccessful path is not enough to answer "no"

34.2 Polynomial-time verification

- Nondeterministic algorithm
  - Magic dice view

    With a magic dice that always makes the right choice at each decision point, a yes-instance can be solved in polynomial time.
  - Verification view
    - If you tell me that an instance is a yes-instance and give me a certificate, I can verify it in polynomial time.

      For example, for the KNAPSACK problem, a successful path can be verified in polynomial time.

34.2 Polynomial-time verification

- Nondeterministic algorithm
  - Verification view (Cont'd)
    - To capture this view, Book introduces verification algorithms.
    - A verification algorithm A verifies x (i.e. verifies that x is a yes-instance) if there is a certificate y such that $A(x, y) = 1$
    - **DEFINITION**

      A verification algorithm A ***verifies*** a language

      $L(A) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \text{such that } A(x, y) = 1\}$

      $=$ the language of all yes instances

34.2 Polynomial-time verification

- The complexity class NP
  - The class NP may be defined in two ways:

$$\text{NP} = \left\{ L \;\middle|\; \begin{array}{c} L \text{ is verified by a polynomial-time} \\ \text{verification algorithm} \end{array} \right\}$$

$$= \left\{ L \;\middle|\; \begin{array}{c} L \text{ is accepted by a polynomial-time} \\ \text{nondeterministic algorithm} \end{array} \right\}$$

NOT decided – the time spent for a no-instance hasn't even been defined!

  -

P	NP	
deterministic	nondeterministic	verification (deterministic)
polynomial time	polynomial time	polynomial time
decide/accept	accept	verify

34.2 Polynomial-time verification

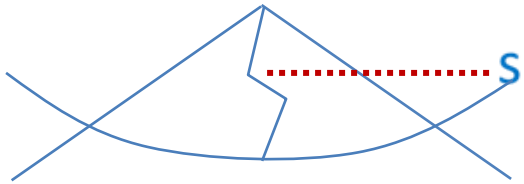
- The complexity class NP
  - **THEOREM** (Exercise 34.2-5)

$$\text{NP} \subseteq \left\{ L \;\middle|\; \begin{array}{c} L \text{ is decided by an exponential–time} \\ \text{deterministic algorithm} \end{array} \right\}$$

Proof

Suppose the nondeterministic algorithm takes at most $p(n)$ time, for some polynomial $p(n)$, to accept a yes-instance.

Then, the length of the shortest successful path $\leq p(n)$.



shortest successful path, length $\leq p(n)$

34.2 Polynomial-time verification

- The complexity class NP
  - **THEOREM** (Cont'd)

    Thus, the deterministic algorithm needs only search each path up to length $p(n)$ and ***decides*** if an instance is a yes-instance by looking for a successful path of length $\leq p(n)$.

    As to the complexity, suppose

    the # of choices at each decision point $\leq r(n)$

    Then, there are at most $r(n)^{p(n)}$ paths of length $\leq p(n)$.

    Since each path takes at most $p(n)$ time to search, the total time is at most $p(n) \cdot r(n)^{p(n)}$.

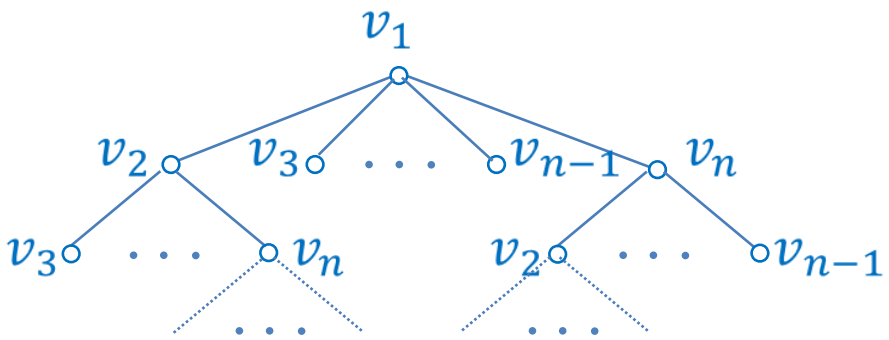
34.2 Polynomial-time verification

- The complexity class NP
  - Comment on $r(n)$
    - For the KNAPSACK problem, $r(n) = 2$
    - Consider the Hamiltonian cycle problem

      Does the undirected graph G have a Hamiltonian cycle,

      i.e. a simple cycle that contains every vertex in G?

      For the backtracking algorithm, $r(n) = n - 1$



34.2 Polynomial-time verification

- The complexity class NP
  - The equality doesn't hold.
  - Example

    HANOI ≡ Does $(\text{move}_1, \ldots, \text{move}_k)$ correctly move n disks?

    THEOREM It takes at least $2^n - 1$ moves to move n disks.

    According to the theorem, the length of a certificate $\geq 2^n - 1$ and thus can't be verified in polynomial time.

    So, HANOI $\notin$ NP

    On the other hand, HANOI can be solved by an exponential-time deterministic algorithm – simply generate a solution and compare it to $(\text{move}_1, \ldots, \text{move}_k)$.

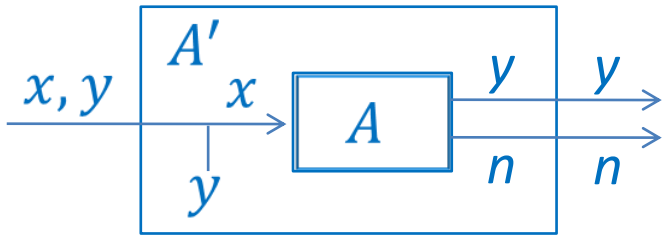
34.2 Polynomial-time verification

- The complexity class NP
  - **THEOREM** P ⊆ NP

    Proof Show that $L \in \text{P} \Rightarrow L \in \text{NP}$

    Suppose that algorithm A decides L in polynomial time.

    Construct a verification algorithm A' by simply ignoring the certificate y and letting $A'(x, y) = 1$ iff $A(x) = 1$



    Then, algorithm A' verifies L in polynomial time.
  - Open problem: Is P = NP ?

34.2 Polynomial-time verification

- The complexity class co-NP
  - co-NP $= \{\bar{L} \mid L \in \text{NP}\}$
  - Open problem: Is NP = co-NP ?

    It seems unlikely.

    For example, KANPSACK $\in$ NP $\Rightarrow$ $\overline{\text{KANPSACK}} \in$ co-NP

    Does $\overline{\text{KANPSACK}} \in$ NP ?

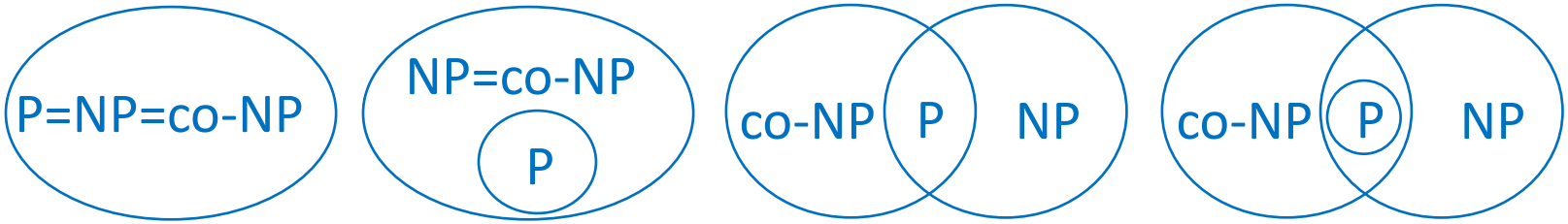
    If it is, then for a yes-instance (i.e. there is no solution to the KANPSACK problem), how can we verify it in polynomial time – after all, if we do not search the entire tree, how can we be sure that there is no solution?

34.2 Polynomial-time verification

- Complexity classes P, co-P, NP, co-NP
  - P = co-P
  - P ⊆ NP
  - co-P ⊆ co-NP

    $\because L \in$ co-P $\Rightarrow \bar{L} \in$ P $\Rightarrow \bar{L} \in$ NP $\Rightarrow L \in$ co-NP
  - P ⊆ NP ∩ co-NP
  - If P = NP, then NP = co-NP (Ex 34.2-10)

    ∵ P = NP ⇒ co-P = co-NP ⇒ P = co-P = co-NP = NP



34.3 NP-completeness and reducibility

- Reducibility
  - Let $L_1 \subseteq \Sigma^*, L_2 \subseteq \Sigma^*$

    L_1 is ***polynomial-time reducible*** to L_2, written as $L_1 \leq_p L_2$

    if $\exists f: \Sigma^* \to \Sigma^*$ such that

    1 f is polynomial-time computable

    2 $x \in L_1 \Leftrightarrow f(x) \in L_2$
  - $L_1 \leq_p L_2$ means L_1 is ***no harder (perhaps, easier) than*** L_2,

    or, as the lemma on the next page illustrates,

    if L_2 is easy, so is L_1 (i.e. $L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$)

    However, it is possible that L_2 is hard, but L_1 is easy.

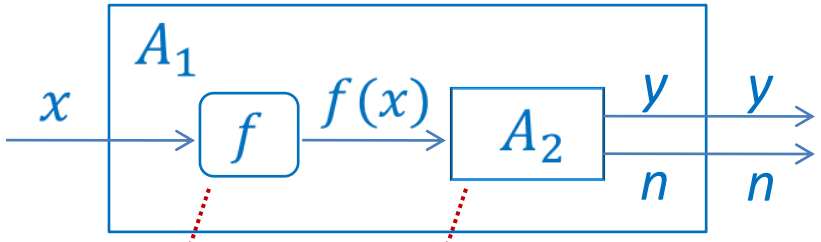
34.3 NP-completeness and reducibility

- Reducibility
  - **THEOREM**

    If $L_1 \leq_p L_2$, then $L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$

    Proof

    Let A_2 decide L_2. Construct A_1 to decide L_1, as follows.



$O(|x|^c)$ $O(|f(x)|^k) = O(|x|^{ck})$, since $|f(x)| = O(|x|^c)$

Thus, A_1 takes a time in

$O(|x|^c) + O(|x|^{ck}) = O(|x|^{\max\{c,ck\}})$

34.3 NP-completeness and reducibility

- The complexity class NPC
  - NPC $= \{L \mid L$ is NP-complete$\}$
  - A language L is NP-complete (NPC) if

    1 $L \in$ NP

    2 $\forall L' \in$ NP, $L' \leq_p L$
  - That is, NPC problems are the hardest problems in NP.
  - A language L is NP-hard if $\forall L' \in$ NP, $L' \leq_p L$

    In particular, $\forall L' \in$ NPC, $L' \leq_p L$
  - That is, NP-hard problems needn't belong to NP, but are ***"at least as hard as NPC problems"***.

34.3 NP-completeness and reducibility

- The complexity class NPC
  - **THEOREM**

    If $L \in$ NPC and $L \in$ P then P = NP

    Proof

    Need only show that NP $\subseteq$ P

    Let $L' \in$ NP

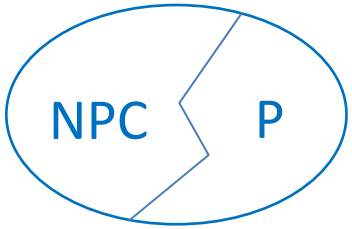
    Then, L is NPC $\Rightarrow L' \leq_p L$

    Thus, $L' \leq_p L$ and $L \in$ P $\Rightarrow L' \in$ P
  - **COROLLARY**

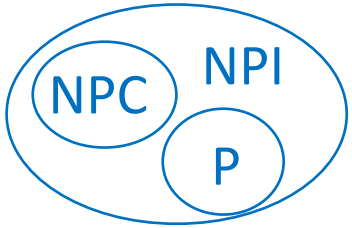
    If $L \in$ NPC and P $\neq$ NP then $L \notin$ P

34.3 NP-completeness and reducibility

- The complexity class NPC
  - Thus, if P ≠ NP, there are two possibilities.



NP (1)



NP (2)

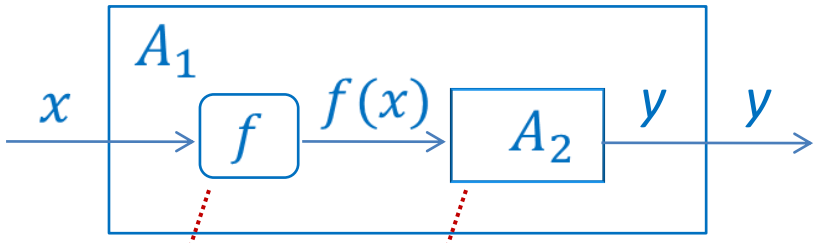
  - It can be shown that if P ≠ NP, then (2) holds.
    - P consists of the easiest problems in NP
    - NPC consists of the hardest problems in NP
    - NPI consists of the "intermediate hardness" problems in NP

34.3 NP-completeness and reducibility

- The complexity class NPC
  - **LEMMA A**

    If $L_1 \leq_p L_2$, then $L_2 \in \text{NP} \Rightarrow L_1 \in \text{NP}$

    Proof



$O(|x|^c)$ $O(|f(x)|^k) = O(|x|^{ck})$, since $|f(x)| = O(|x|^c)$

Nondeterministic algorithm A_2 accepts yes-instances in polynomial time, so is nondeterministic algorithm A_1

34.3 NP-completeness and reducibility

- The complexity class NPC
  - **LEMMA B** $L_1 \leq_p L_2$ iff $\bar{L}_1 \leq_p \bar{L}_2$

    $L_1 \leq_p L_2 \Leftrightarrow x \in L_1$ iff $f(x) \in L_2$

    $\Leftrightarrow x \notin L_1$ iff $f(x) \notin L_2$

    $\Leftrightarrow x \in \bar{L}_1$ iff $f(x) \in \bar{L}_2$

    $\Leftrightarrow \bar{L}_1 \leq_p \bar{L}_2$
  - **LEMMA C** If $L \in$ NPC and $\bar{L} \in$ NP, then NP = co-NP

    NP $\subseteq$ co-NP

    $L' \in \text{NP} \Rightarrow L' \leq_p L$ $\quad \because L \in \text{NPC}$

    $\Rightarrow \bar{L}' \leq_p \bar{L}$ $\quad \because$ lemma B

    $\Rightarrow \bar{L}' \in \text{NP}$ $\quad \because \bar{L} \in$ NP and lemma A

    $\Rightarrow L' \in$ co–NP

34.3 NP-completeness and reducibility

- The complexity class NPC
  - **LEMMA C** If $L \in$ NPC and $\bar{L} \in$ NP, then NP = co-NP

    co-NP $\subseteq$ NP

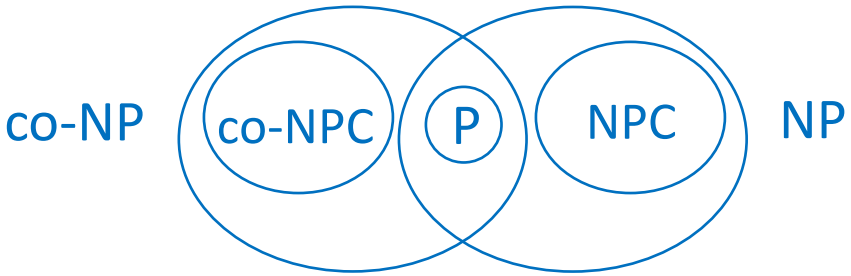
    $L' \in$ co–NP $\Rightarrow \bar{L}' \in$ NP

    $\Rightarrow \bar{L}' \leq_p L$ $\quad \because L \in$ NPC

    $\Rightarrow L' \leq_p \bar{L}$ $\quad \because$ lemma B

    $\Rightarrow L' \in$ NP $\quad \because \bar{L} \in$ NP and lemma A
  - **COROLLARY**

    If NP $\neq$ co-NP, then $L \in$ NPC $\Rightarrow \bar{L} \notin$ NP $\Rightarrow L \notin$ co–NP



34.3 NP-completeness and reducibility

- The complexity class NPC
  - **DEFINITION** (Ex 34.3-6)

    Let C be a class of languages, a language L is C-complete if

    1 $L \in C$

    2 $\forall L' \in C, L' \leq_p L$
  - **LEMMA D** (Ex 34.3-7)

    L is NPC iff $\bar{L}$ is co-NPC

    Proof

    $L \in \text{NPC} \Leftrightarrow \forall L' \in \text{NP}, L' \leq_p L$

    $\Leftrightarrow \forall \bar{L}' \in \text{co-NP}, \bar{L}' \leq_p \bar{L}$ $\because$ lemma B

    $\Leftrightarrow \bar{L} \in \text{co-NPC}$

34.3 NP-completeness and reducibility

- The complexity class NPC
  - **LEMMA E**

    If $L \in$ co-NPC and $\bar{L} \in$ co-NP, then NP = co-NP

    Proof

    $L \in$ co-NPC $\Rightarrow \bar{L} \in$ NPC $\because$ lemma D

    $\bar{L} \in$ co-NP $\Rightarrow L \in$ NP

    It follows lemma C that NP = co-NP
  - **COROLLARY**

    If NP $\neq$ co-NP, then $L \in$ co-NPC $\Rightarrow \bar{L} \notin$ co-NP $\Rightarrow L \notin$ NP

34.4 NP-completeness proofs

- NP-completeness proofs
  - **LEMMA** (Ex 34.3-2)

    $\leq_p$ is transitive, i.e. $L \leq_p L_1$ and $L_1 \leq_p L_2 \Rightarrow L \leq_p L_2$
  - **THEOREM**

    Let $L_1 \in$ NPC, then

    $L_2 \in$ NP and $L_1 \leq_p L_2 \Rightarrow L_2 \in$ NPC

    Proof

    $L_1 \in \text{NPC} \Rightarrow \forall L \in \text{NP}, L \leq_p L_1$

    $\Rightarrow \forall L \in \text{NP}, L \leq_p L_2 \quad \because L_1 \leq_p L_2$ and transitive

    $\Rightarrow L_2 \in$ NPC

34.4 NP-completeness proofs

- The first NPC problem
  - A Boolean formula is in CNF (Conjunctive Normal Form) if it is of the form $\wedge(\vee l_i)$, where l_i is a literal.
    N.B. A literal is a variable or the negation of a variable.
  - A Boolean formula is ***satisfiable*** if it is true under some truth assignment of variables contained in it.
    - $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$ is unsatisfiable.
    - $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$ is satisfiable, as it is true under the truth assignment $x_1 = 1, x_2 = 1$ or $0, x_3 = 1$ or 0.

34.4 NP-completeness proofs

- The first NPC problem
  - CNF-SAT ≡ Is a formula in CNF satisfiable?

    CNF-SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula in CNF$\}$
  - Brute-force approach

    A formula with n variables has 2^n truth assignments.
  - **THEOREM** (Cook, 1971)

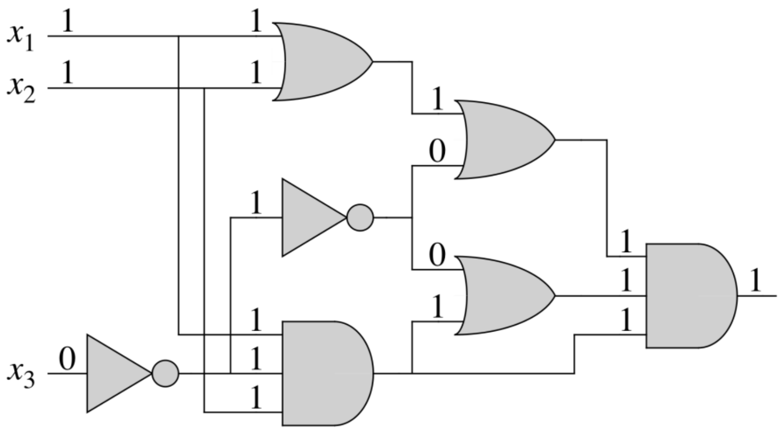
    CNF-SAT is NPC

34.4 NP-completeness proofs

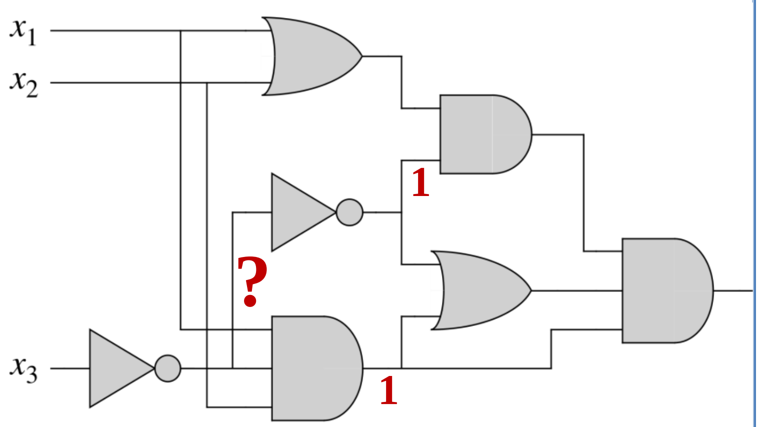
- The first NPC problem
  - Book's 1st NPC problem:

    CIRCUIT-SAT $= \{\langle C \rangle \mid C$ is a satisfiable circuit$\}$
  - A circuit with n input wires has 2^n truth assignments.
  - Example

Satisfiable circuit



Unsatisfiable circuit



34.4 NP-completeness proofs

- The first NPC problem
  - **THEOREM** CIRCUIT-SAT is NPC (Sec. 34.3)

    CIRCUIT-SAT $\in$ NP

    $\because$ A certificate consisting of a satisfying assignment to input wires of a circuit can be verify in linear time by simply running through all the gates in the circuit.

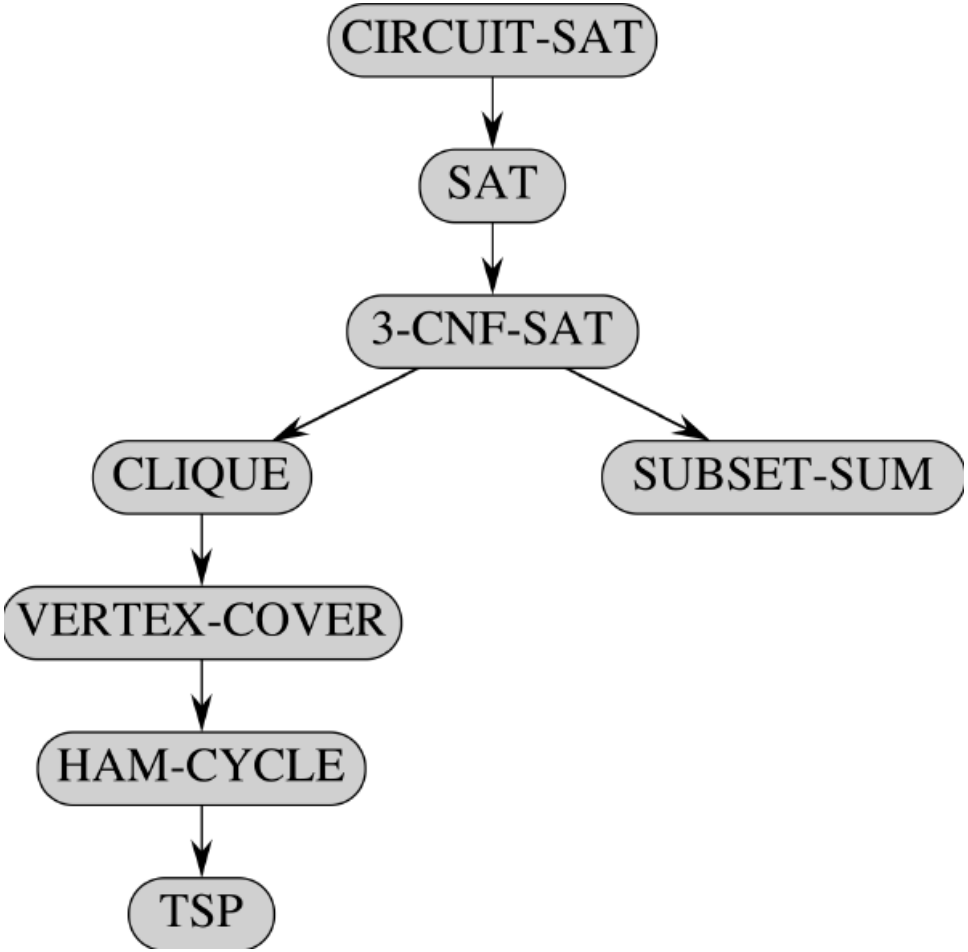
    Note: the size of the circuit = number of gates in it

    $\forall L \in$ NP, $L \leq_p$CIRCUIT-SAT

    Idea: Let A be a nondeterministice algorithm that accepts L. Simply assemble the circuits used to execute A to obtain a circuit for CIRCUIT-SAT.

34.4 NP-completeness proofs

- NPC proofs in Sec. 34.4 and 34.5



34.4 NP-completeness proofs

- Formula satisfiability
  - SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula$\}$
  - **THEOREM** SAT is NPC

    SAT ∈ NP

    ∵ A certificate consisting of a satisfying assignment for a formula can be verified in polynomial time by substitution and evaluation:

    $\phi = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

    Certificate: $x_1 = x_2 = 0, x_3 = x_4 = 1$

    $\phi = ((0 \to 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0$

    $= (1 \vee \neg(1 \vee 1)) \wedge 1$

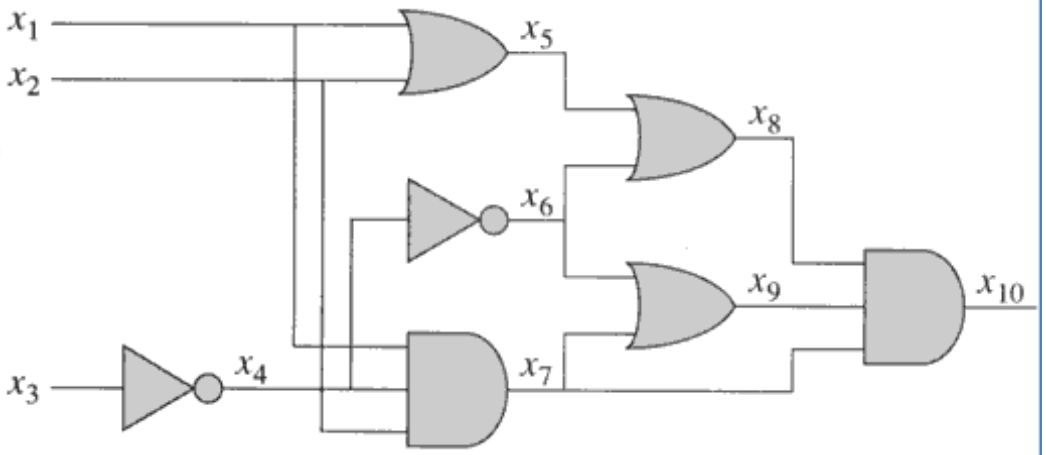
    $= (1 \vee 0) \wedge 1 = 1$

34.4 NP-completeness proofs

- Formula satisfiability
  - **THEOREM** SAT is NPC (Cont'd)

    CIRCUIT-SAT $\leq_p$ SAT

    Reduce the circuit C to a formula ϕ



$$\phi = x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow x_1 \vee x_2) \wedge (x_6 \leftrightarrow \neg x_4)$$
$$\wedge (x_7 \leftrightarrow x_1 \wedge x_2 \wedge x_4) \wedge (x_8 \leftrightarrow x_5 \vee x_6)$$
$$\wedge \underbrace{(x_9 \leftrightarrow x_6 \vee x_7)}_{\text{clause}} \wedge (x_{10} \leftrightarrow x_7 \wedge x_8 \wedge x_9)$$

34.4 NP-completeness proofs

- Formula satisfiability
  - **THEOREM** SAT is NPC (Cont'd)

    Clearly, this reduction is polynomial-time computable, ∵ it introduces one clause per gate.

    Next, we show that C is satisfiable ⇒ ϕ is satisfiable

    C is satisfiable

    ⇒ ∃ truth assignment of x_1, x_2, x_3 that makes $x_{10} = 1$

    Let x_1, x_2, x_3 have the same truth values

    Let $x_4, x_5, \ldots, x_{10}$ have the corresponding wire values in C

    Then, ϕ is true under this truth assignment, since $x_{10} = 1$ and each clause describing the logic of a gate evaluates to 1

34.4 NP-completeness proofs

- Formula satisfiability
  - **THEOREM** SAT is NPC (Cont'd)

    Finally, we show that ϕ is satisfiable $\Rightarrow C$ is satisfiable

    ϕ is satisfiable

    $\Rightarrow \exists$ truth assignment of $x_1, x_2, \ldots, x_{10}$ that makes $\phi = 1$

    $\Rightarrow$ each clause in ϕ is true under this truth assignment

    $\Rightarrow x_{10} = 1$ and each gate functions correctly

    $\Rightarrow C$ is satisfibale under the same truth assignment of x_1, x_2, x_3

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - A formula is in 3-CNF if it is of the form

    $\wedge \underbrace{(l_1 \vee l_2 \vee l_3)}_{\text{clause}}$, where l_i is a literal.
  - 3-CNF-SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable formula in 3-CNF$\}$
  - **THEOREM** 3-CNF-SAT is NPC

    3-CNF-SAT$\in$ NP

    Similar to the proof of SAT $\in$ NP

    SAT $\leq_p$ 3-CNF-SAT

    The reduction can be broken into 4 steps.

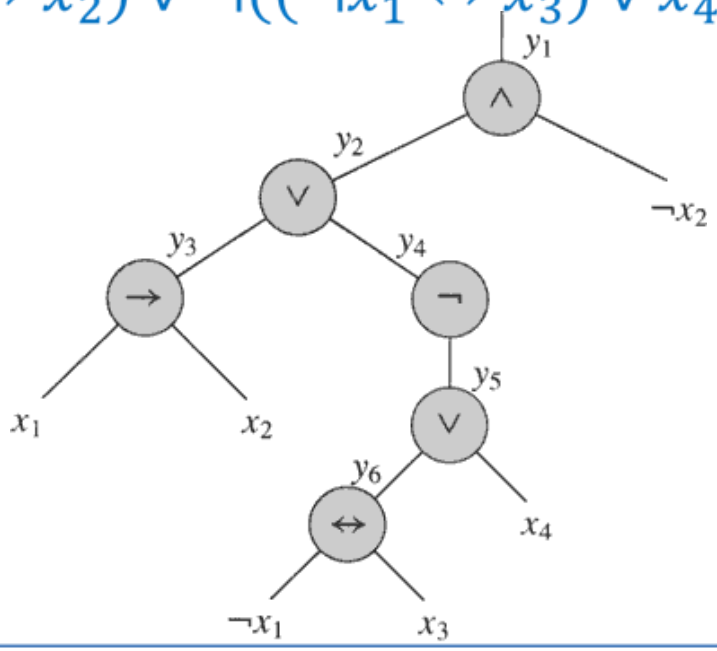
34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

    Step 1: Parse ϕ to a "circuit" (i.e. parse tree) C in which each node has 1 or 2 children, e.g.

    $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

    Circuit C



34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

    This step is polynomial-time computable, ∵ it introduces at most one gate per connective in ϕ.

    ϕ is satisfiable $\Leftrightarrow$ C is satisfiable

    ∵ Both yield the same truth value under the same truth assignment of x_1, x_2, x_3, x_4, since the circuit C is nothing but the graphical representation of the formula ϕ.

    Step 2: Reduce the circuit C to a formula ϕ'

    This step is similar to the reduction CIRCUIT-SAT $\leq_p$ SAT.

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \wedge (y_2 \leftrightarrow y_3 \vee y_4)$$
$$\wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \wedge (y_4 \leftrightarrow \neg y_5)$$
$$\wedge (y_5 \leftrightarrow y_6 \vee x_4) \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))$$

This step polynomial-time computable,∵ it introduces 1 variable and 1 clause per gate in C.

C is satisfiable $\Leftrightarrow$ ϕ' is satisfiable

The proof is similar to that of CIRCUIT-SAT $\leq_p$ SAT.

Let's write $\phi' = \wedge \phi_i'$ where each ϕ_i' is a clause that consists of at most 3 variables

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

    Step 3: Reduce $\phi' = \wedge \phi_i'$ to $\phi'' = \wedge \phi_i''$ by reducing each ϕ_i' to a $\leq$3-CNF formula ϕ_i''

    For example, let $\phi_i' = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$

    Construct the truth table and look at the truth assignments that falsify ϕ_i'

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

$$\phi_i' = \neg((y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2))$$

$$= (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

$$= \phi_i''$$

This step is polynomial-time computable, $\because$ it introduces at most 8 clauses for each clause in ϕ'.

ϕ' is satisfiable $\Leftrightarrow$ ϕ'' is satisfiable

$\because \forall i$, ϕ_i' and ϕ_i'' are logical equivalent $\Rightarrow$ so are ϕ' and ϕ''

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

    Step 4: Reduce $\phi'' = \wedge \phi_i''$ to $\phi''' = \wedge \phi_i'''$ by reducing each clause C_i of ϕ_i'' to a 3-CNF formula D_i of ϕ_i'''

    Case 1: C_i has 3 literals

    Let $D_i = C_i$

    Case 2: C_i has 2 literals, i.e. $C_i = l_1 \vee l_2$

    Let $D_i = (l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$

    Case 3: C_i has 1 literal, i.e. $C_i = l$

    Let $D_i = (l \vee p \vee q) \wedge (l \vee \neg p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee \neg q)$

    where p and q are new variables.

34.4 NP-completeness proofs

- 3-CNF satisfiability
  - **THEOREM** 3-CNF-SAT is NPC (Cont'd)

    This step is polynomial-time computable, ∵ it introduces at most 4 clauses for each clause in ϕ''

    ϕ'' is satisfiable $\Leftrightarrow$ ϕ''' is satisfiable

    ∵ C_i and D_i are logical equivalent $\Rightarrow$ so are ϕ'' and ϕ'''

    In summary:

    Formula $\phi \Rightarrow$ Circuit $C \Rightarrow \wedge \leq$3-variable-clause formula ϕ' $\Rightarrow \leq$3-CNF formula $\phi'' \Rightarrow$ 3-CNF formula ϕ'''

34.5 NP-completeness problems

- The clique problem
  - Let $G = (V, E)$ be an undirected graph

    $V' \subset V$ is a clique of G if $\forall u, v \in V', (u, v) \in E$.

    i.e. V' is the set of the vertices of a complete subgraph of G
  - Every single vertex is a clique of size 1; so, find a clique of maximum size.
  - Clique decision problem

    Version 1: Given G and k, is there a click of size $\geq k$?

    Version 2: Given G and k, is there a click of size $= k$?

    These two versions are equivalent.

    Book considers version 2.

34.5 NP-completeness problems

- The clique problem
  - Naïve algorithm

    For each k-subset of V, check if it is a clique.
  - Let $n = |V|$, this naïve algorithm takes a time in $\Omega\left(k^2\binom{n}{k}\right)$

    Since $k \leq n$, the k^2 term is polynomial in $|\langle G, k\rangle| = O(n^2) + \lg k$

    On the other hand, the $\binom{n}{k}$ term is polynomial for small k, but is exponential as $k \to n/2$, since

$$\binom{n}{n/2} = \frac{n!}{(n/2)!\,(n/2)!} = \Omega\left(2^{n/2}\right)$$

34.5 NP-completeness problems

- The clique problem
  - CLIQUE $= \{\langle G, k\rangle \mid$ Graph G has a clique of size $k\}$
  - **THEOREM** CLIQUE is NPC

    CLIQUE $\in$ NP $\because$ A certificate can be verified in $O(k^2)$ time.

    3-CNF-SAT $\leq_p$ CLIQUE

    Let $\phi = C_1 \wedge C_1 \wedge \cdots \wedge C_k$ where $C_r = l_1^r \vee l_2^r \vee l_3^r$

    Reduce ϕ to the graph $G = (V, E)$ as follows:

    $V = \{v_1^r, v_2^r, v_3^r \mid 1 \leq r \leq k\}$, v_i^r corresponds to l_i^r

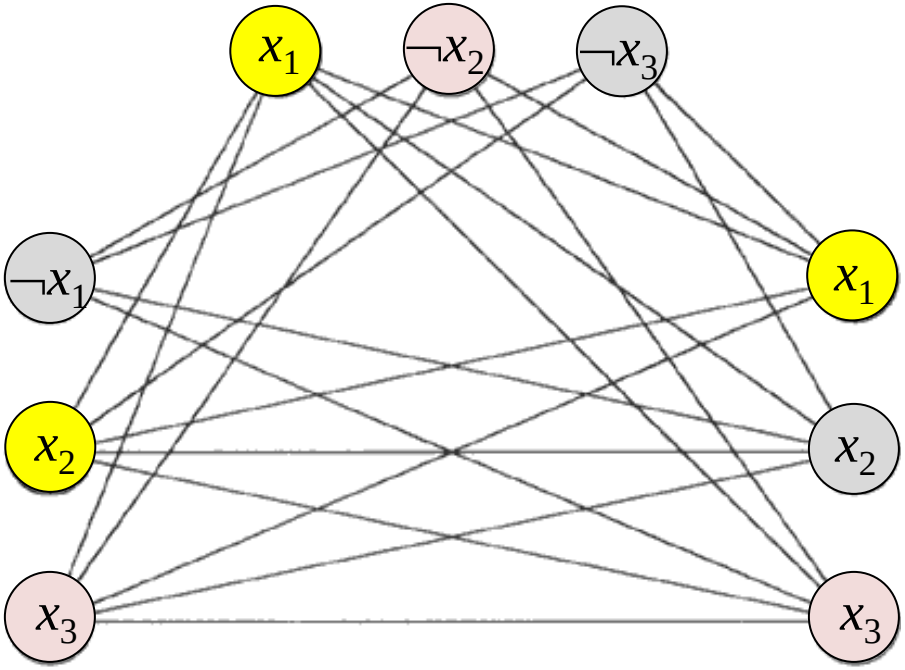
    $E = \left\{(v_i^r, v_j^s) \,\middle|\, r \neq s, l_i^r \neq \neg l_j^s\right\}$

    Since $|V| = 3k$ and $|E| = O(k^2)$, the graph $G = (V, E)$ can be constructed in polynomial time.

34.5 NP-completeness problems

- The clique problem
  - **THEOREM** CLIQUE is NPC (Cont'd)

$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$



$C_2 = \neg x_1 \vee x_2 \vee x_3$

$C_3 = x_1 \vee x_2 \vee x_3$

34.5 NP-completeness problems

- The clique problem
  - **THEOREM** CLIQUE is NPC (Cont'd)

    Satisfying assignment $x_1 = x_2 = 1, x_3 =$ don't care

    $\Leftrightarrow$ Clique with yellow-colored vertices.

    Satisfying assignment $x_1 =$ don't care, $x_2 = 0, x_3 = 1$

    $\Leftrightarrow$ Clique with pink-colored vertices.

    In general, ϕ is satisfiable $\Rightarrow$ G has a clique of size k

    $\because \phi$ is satisfiable $\Rightarrow \forall r \exists l_i^r \in C_r$ such that $l_i^r = 1$

    Let $V' = \{v_i^r \mid$ for each r, pick one $l_i^r = 1, 1 \le r \le k\}$

    Then, V' is a clique of size k.

34.5 NP-completeness problems

- The clique problem
  - **THEOREM** CLIQUE is NPC (Cont'd)

    $\because \forall v_i^r, v_j^s \in V', r \neq s$

    $l_i^r = l_j^s = 1 \Rightarrow l_i^r \neq \neg l_j^s \Rightarrow (v_i^r, v_j^s) \in E$

    Conversely, G has a clique V' of size $k \Rightarrow \phi$ is satisfiable

    $\because |V'| = k \Rightarrow$ exactly one of $v_1^r, v_2^r, v_3^r \in V'$, for each r

    Let $l_i^r = 1$, if $v_i^r \in V'$

    Literals not assigned a truth value are don't care. This truth assignment is consistent, i.e. without assigning 1 to both a literal and its complement, $\because$ no edge between them.

    Then, C_r is satisfiable for all $r \Rightarrow \phi$ is satisfiable

34.5 NP-completeness problems

- The vertex cover problem
  - Let $G = (V, E)$ be an undirected graph
    $V' \subset V$ is a vertex cover of G if $\forall (u, v) \in E$, $u \in V'$ or $v \in V'$ or both
  - The set V itself is a vertex cover; so, find a vertex cover of minimum size .
  - Vertex cover decision problem
    Ver 1: Given G and k, is there a vertex cover of size $\leq k$?
    Ver 2: Given G and k, is there a vertex cover of size $= k$?
    These two versions are equivalent.
    Book considers version 2.

34.5 NP-completeness problems

- The vertex cover problem
  - Naïve algorithm

    For each k-subset of V, check if it is a vertex cover
  - Let $n = |V|$, this naïve algorithm takes $\Omega\left(|E|\binom{n}{k}\right)$ time in the worst case, which is exponential as $k \to n/2$
  - VERTEX-COVER

    $= \{\langle G, k\rangle \mid$ Graph G has a vector cover of size $k\}$
  - **THEOREM** VERTEX-COVER is NPC

    VERTEX-COVER $\in$ NP

    $\because$ A certificate can be verified in $O(k|E|)$ time.

34.5 NP-completeness problems

- The vertex cover problem
  - **THEOREM** VERTEX-COVER is NPC (Cont'd)

    CLIQUE $\leq_p$ VERTEX-COVER

    Reduce the CLIQUE instance $\langle G, k \rangle$, where $G = (V, E)$, to a VERTEX-COVER instance $\langle \bar{G}, |V| - k \rangle$, where $\bar{G} = (V, \bar{E})$

    The reduction can be done in $O(|V|^2)$ time.

    G has a clique of size $k \Rightarrow \bar{G}$ has a vector cover of size $|V| - k$

    Suppose G has a clique $V' \subseteq V$ of size k

    CLAIM $V - V'$ is a vertex cover of $\bar{G}$ of size $|V| - k$

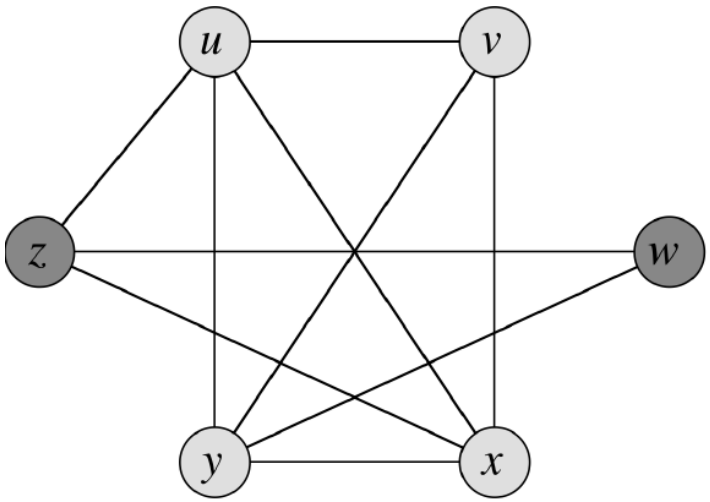
    $(v, w) \in \bar{E} \Rightarrow (v, w) \notin E$

    $\Rightarrow v \notin V' \vee w \notin V'$ $\quad \because V'$ is a clique of G

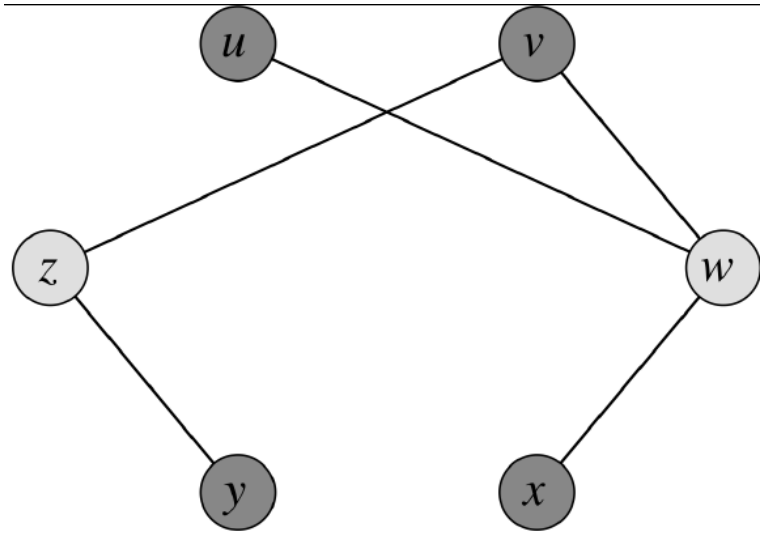
    $\Rightarrow v \in V - V' \vee w \in V - V'$ i.e. edge is coved

34.5 NP-completeness problems

- The vertex cover problem
  - **THEOREM** VERTEX-COVER is NPC (Cont'd)



$\langle G, 4 \rangle$ clique $\{u, v, x, y\}$



$\langle \bar{G}, 2 \rangle$ vertex cover $\{w, z\}$

34.5 NP-completeness problems

- The vertex cover problem
  - **THEOREM** VERTEX-COVER is NPC (Cont'd)

    $\bar{G}$ has a vertex cover of size $|V| - k \Rightarrow G$ has a clique of size k

    Suppose $\bar{G}$ has a vertex cover $V' \subseteq V$ of size $|V| - k$

    Claim $V - V'$ is a clique of G of size k

    $v, w \in V - V'$

    $\Rightarrow v, w \notin V'$

    $\Rightarrow (v, w) \notin \bar{E}$ $\quad \because V'$ is a vertex cover of $\overline{G}$

    $\Rightarrow (v, w) \in E$ $\quad$ i.e. edge exists

34.5 NP-completeness problems

- The hamiltonian-cycle problem
  - Let $G = (V, E)$ be an undirected graph

    A hamiltonian cycle of G is a simple cycle that contains every vertex in G.
  - A hamiltonian graph is a graph that contains a hamiltonian cycle.
  - Is a given G hamiltonian?

    Naïve algorithm

    For each permutation of vertices of G, check if it is a cycle.

    This takes a times in $\Omega(n!)$ which is exponential in $|\langle G \rangle| = O(n^2)$, where $n = |V|$.

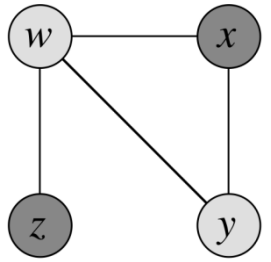
34.5 NP-completeness problems

- The hamiltonian-cycle problem
  - HAM-CYCLE $= \{\langle G \rangle \mid G \text{ is a hamiltonian graph}\}$
  - **THEOREM** HAM-CYCLE is NPC

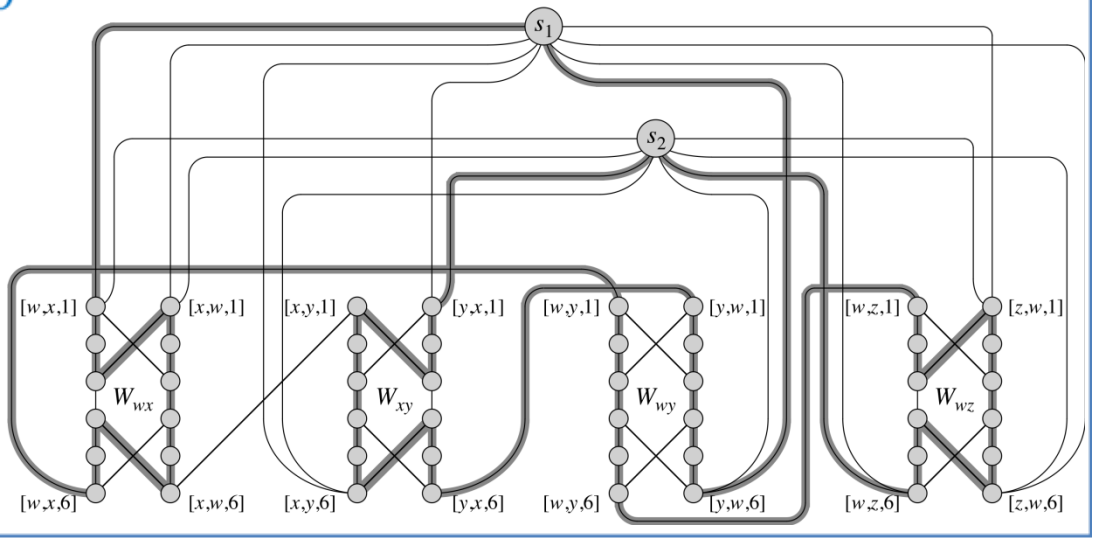
    HAM-CYCLE $\in$ NP

    $\because$ A certificate can be verified in $O(|V|)$ time.

    VERTEX-COVER $\leq_p$ HAM-CYCLE



$\Rightarrow$



SKIP

34.5 NP-completeness problems

- The traveling-salesman problem
  - Let $G = (V, E)$ be an undirected, weighted ***complete*** graph

    $c: V \times V \to \mathbf{Z}$ is the cost function

    Find a ***tour***, i.e. a hamiltonian cycle, with minimum cost
  - Naïve algorithm

    Find the minimum cost among $(|V| - 1)!/2$ possible tours.
  - TSP $= \{\langle G, c, k\rangle \mid G$ has a tour with cost $\leq k\}$
  - **THEOREM** TSP is NPC

    TSP $\in$ NP

    $\because$ A certificate can be verified in $O(|V|)$ time.

34.5 NP-completeness problems

- The traveling-salesman problem
  - **THEOREM** TSP is NPC (Cont'd)

    HAM-CYCLE $\leq_p$ TSP

    Reduce the HAM-CYCLE instance $G = (V, E)$ to a TSP instance $\langle G', c, 0 \rangle$, where $G' = (V, E')$ is the complete graph, and

    $$c(i,j) = \begin{cases} 0, & \text{if } (i,j) \in E \\ 1, & \text{if } (i,j) \notin E \end{cases}$$

    Clearly, this reduction can be done in $O(|V|^2)$ time

    G has a hamiltonian cycle $\Rightarrow$ G' has a tour with cost ≤ 0

34.5 NP-completeness problems

- The traveling-salesman problem
  - **THEOREM** TSP is NPC

    $\because$ G has a hamiltonian cycle h

    $\Rightarrow$ each edge in $h \in E$

    $\Rightarrow$ the cost of each edge in $h = 0$

    $\Rightarrow$ h is a tour of G' with cost 0, which is ≤ 0

    G' has a tour with cost $\leq 0 \Rightarrow G$ has a hamiltonian cycle

    $\because$ G' has a tour h with cost ≤ 0

    $\Rightarrow$ the tour h must have cost $= 0$

    $\Rightarrow$ the cost of each edge in $h = 0$

    $\Rightarrow$ each edge in $h \in E$

    $\Rightarrow$ h is a hamiltonian cycle of G

34.5 NP-completeness problems

- The subset-sum problem
  - Given $S \subset \mathbf{N}$ and a target $t \in \mathbf{N}$

    Is there a subset $S' \subseteq S$ whose elements sum to t?
  - Naïve algorithm

    Check all of $2^{|S|}$ subsets
  - SUBSET-SUM

    $= \{\langle S, t\rangle \mid S \text{ has a subset } S' \text{such that } t = \sum_{s \in S'} s\}$
  - **THEOREM** SUBSET-SUM is NPC

    SUBSET-SUM $\in$ NP

    $\because$ A certificate can be verified in $O(|S|)$ time.

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)

    3-CNF-SAT $\leq_p$ SUBSET-SUM

    $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

    $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$

    $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

    $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$

    $C_4 = (x_1 \vee x_2 \vee x_3)$

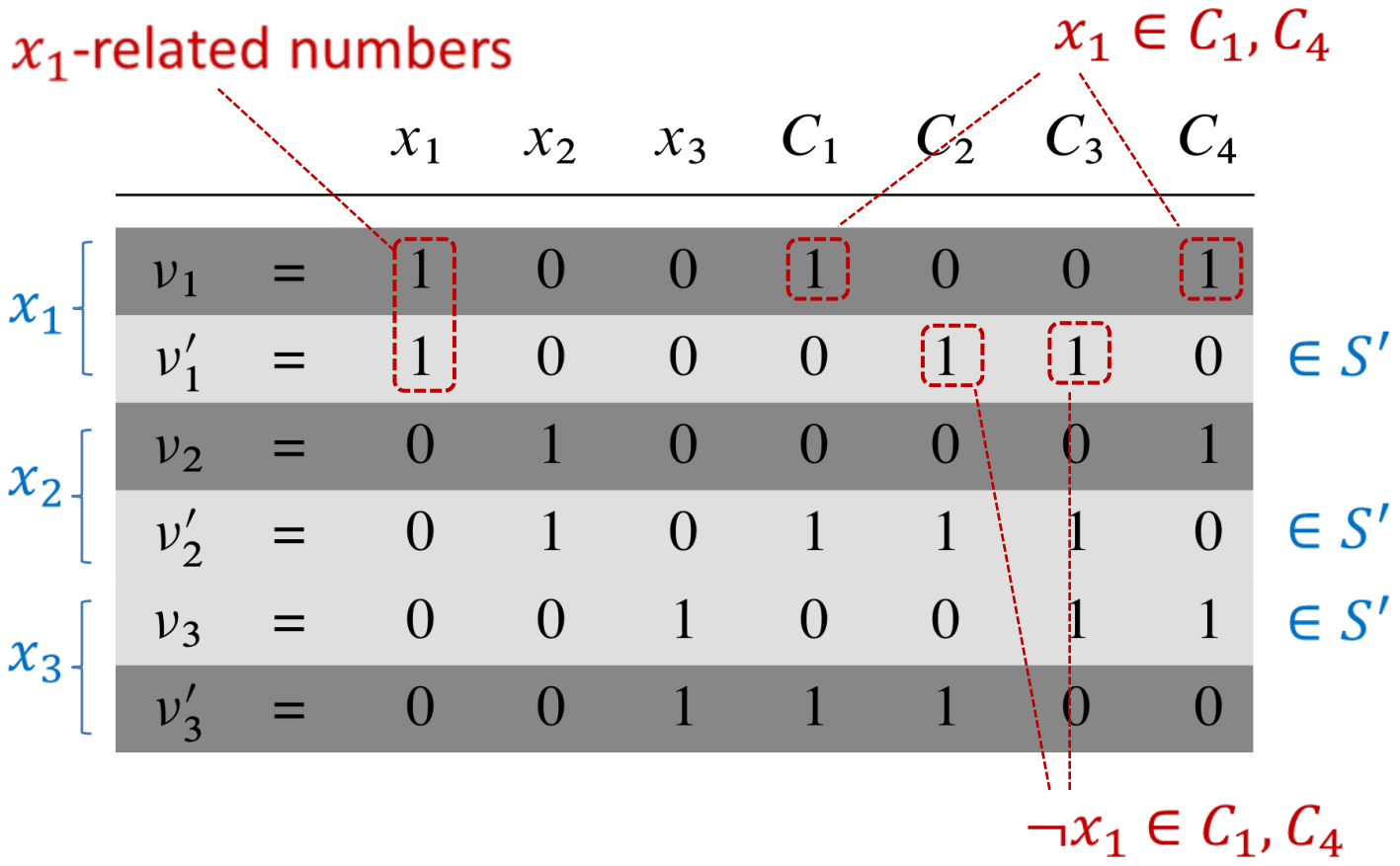
    An satisfying assignment $x_1 = 0, x_2 = 0, x_3 = 1$.

    There are 3 variables and 4 clauses. We shall create

    $2(3 + 4) + 1$ numbers for S and t, each having 7 digits.

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)



		x_1	x_2	x_3	C_1	C_2	C_3	C_4	
x_1	v_1 =	1	0	0	1	0	0	1	
	v_1' =	1	0	0	0	1	1	0	$\in S'$
x_2	v_2 =	0	1	0	0	0	0	1	
	v_2' =	0	1	0	1	1	1	0	$\in S'$
x_3	v_3 =	0	0	1	0	0	1	1	$\in S'$
	v_3' =	0	0	1	1	1	0	0	

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)

C_1-related numbers

			x_1	x_2	x_3	C_1	C_2	C_3	C_4	
C_1	s_1	=	0	0	0	1	0	0	0	$\in S'$
	s_1'	=	0	0	0	2	0	0	0	$\in S'$
C_2	s_2	=	0	0	0	0	1	0	0	
	s_2'	=	0	0	0	0	2	0	0	$\in S'$
C_3	s_3	=	0	0	0	0	0	1	0	$\in S'$
	s_3'	=	0	0	0	0	0	2	0	
C_4	s_4	=	0	0	0	0	0	0	1	$\in S'$
	s_4'	=	0	0	0	0	0	0	2	$\in S'$
	t	=	1	1	1	4	4	4	4	

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)

    In general, if ϕ has n variables and k clauses, then S and t have $2(n+k)+1$ numbers, each having $n+k$ digits, which can be created in polynomial time

    ϕ is satisfiable

    $\Rightarrow S$ has a subset S' whose elements sum to t

    Given a satisfying assignment of ϕ, the set S' contains
    - one number related to each variable x_i
    - one or two numbers related to each clause C_i

    as described below.

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)
    - One number related to each variable x_i

      If $x_i = 1$, let $v_i \in S'$

      If $x_i = 0$, let $v_i' \in S'$

      The x_i-labeled digit of any number unrelated to x_i is 0

      $\Rightarrow$ exactly one number in S' whose x_i-labeled digit is 1

      $\Rightarrow$ the sum of all x_i-labeled digits of numbers in S'

      $= 1$

      $=$ the x_i-labeled digit of t

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)
    - One or two numbers related to each clause C_i

      ϕ is satisfiable $\Rightarrow$ each $C_i = 1$

      $\Rightarrow$ the sum of C_i-labeled digits of the numbers included above $= 1, 2,$ or 3 $\because$ 3-CNF

      If sum $= 1$, let $s_i, s_i' \in S'$

      If sum $= 2$, let $s_i' \in S'$

      If sum $= 3$, let $s_i \in S'$

      Then, the sum of all C_i-labeled digits of numbers in S' $= 4 =$ the C_i-labeled digit of t

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)

    S has a subset S' whose elements sum to t

    $\Rightarrow \phi$ is satisfiable

    Given S' whose elements sum to t

    Since the sum of all x_i-labeled digits of numbers in $S' = 1$,

    either $v_i \in S'$ or $v_i' \in S'$, but not both.

    If $v_i \in S'$, let $x_i = 1$

    If $v_i' \in S'$, let $x_i = 0$

34.5 NP-completeness problems

- The subset-sum problem
  - **THEOREM** SUBSET-SUM is NPC (Cont'd)

    CLAIM Each $C_k = 1$ under this truth assignment.

    Since the sum of all C_k-labeled digits of numbers in $S' = 4$,

    there is a $v_i \in S'$ or $v_i' \in S'$ whose C_k-labeled digit $= 1$

    If $v_i \in S'$ and its C_k-labeled digit $= 1$

    $\Rightarrow x_i$ appears in C_k

    $\Rightarrow C_k = 1 \quad \because x_i = 1$

    If $v_i' \in S'$ and its C_k-labeled digit $= 1$

    $\Rightarrow \neg x_i$ appears in C_k

    $\Rightarrow C_k = 1 \quad \because x_i = 0$

Supplementary: NP-Hard

- Karp reducibility $\leq_p$
  - Polynomial-time reduction (or transformation)
  - For decision problems only
  - $L_1 \leq_p L_2$

    Algorithm A_1 for L_1 may call algorithm A_2 for L_2 once
  - **DEFINITION** (Mentioned before)

    A decision problem L is NP-hard

    if $\forall L' \in$ NP, $L' \leq_p L$ (or, $\exists L' \in$ NPC, $L' \leq_p L$)

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - Polynomial-time Turing reduction
  - For decision and optimization problems
  - $R_1 \leq_T R_2$

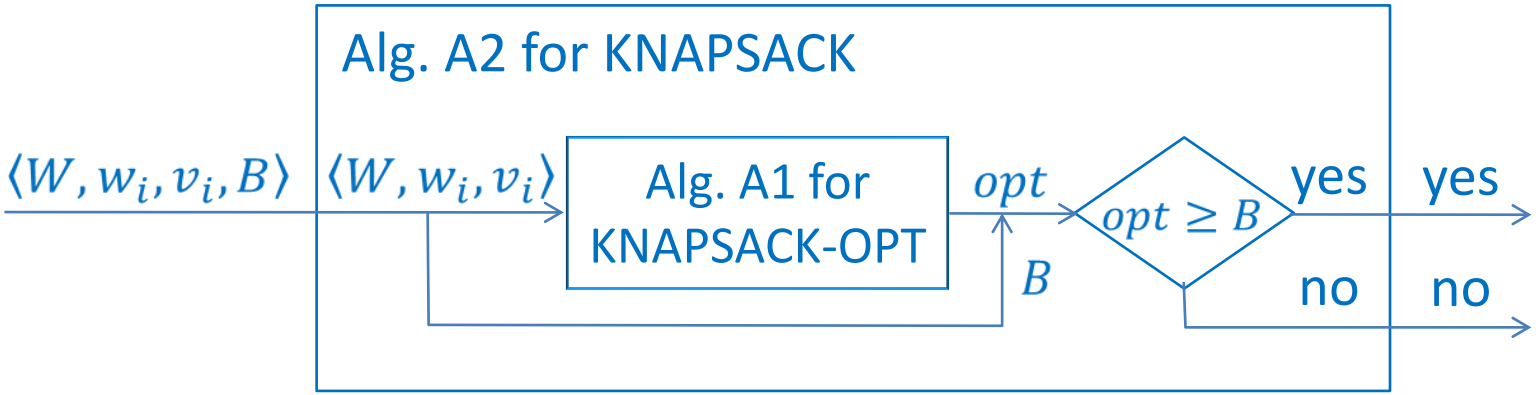
    Algorithm A_1 for R_1 may call algorithm A_2 for R_2 many times, as long as the total time is polynomial
  - NP-hard may be extended to optimization problems.
  - **DEFINITION**

    A decision or optimization problem R is NP-hard if $\forall L \in \text{NP}, L \leq_T R$ (or, $\exists L \in \text{NPC}, L \leq_T R$)

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** KANPSACK-OPT is NP-hard

    KANPSACK $\leq_T$ KANPSACK-OPT



The reduction takes $O(1)$ time.

Note that this reduction doesn't use the full power of Cook reduction, since algorithm A2 calls algorithm A1 only once.

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - The following illustrates the full power of Cook reduction.
  - **THEOREM** KANPSACK-OPT can be solved in polynomial time iff KANPSACK $\in$ P.

    $\Rightarrow$ KANPSACK $\leq_T$ KANPSACK-OPT

    Since KANPSACK-OPT can be solved in polynomial time, Algorithm A1 takes a time in $O(|\langle W, v_i, w_i\rangle|^c)$, $c > 0$

    Since $|\langle W, v_i, w_i\rangle| < |\langle W, v_i, w_i, B\rangle|$, Algorithm A2 takes a time in

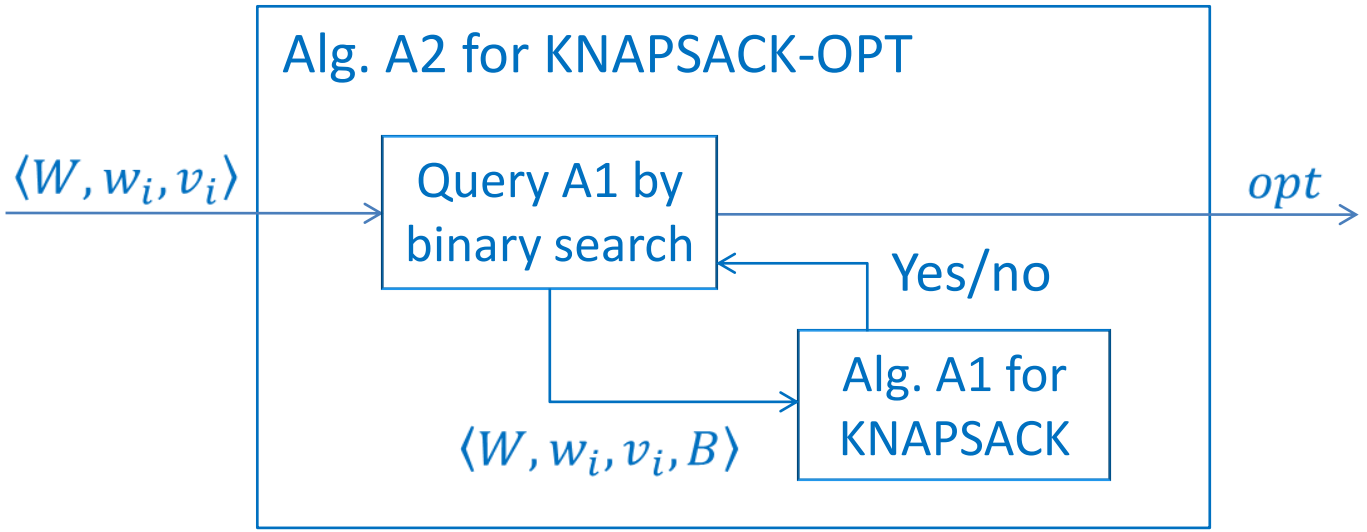
    $O(|\langle W, v_i, w_i\rangle|^c) + O(1) = O(|\langle W, v_i, w_i, B\rangle|^c)$

    which is a polynomial in terms of input size $|\langle W, v_i, w_i, B\rangle|$

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** (Cont'd)

    $\Leftarrow$ KANPSACK-OPT $\leq_T$ KANPSACK



Algorithm A2 uses binary search to query Algorithm A1 as follows:

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** (Cont'd)

    Algorithm A2

    min $= 0$, max $= 1 + \sum v_i$

    while max $-$ min $\neq 1$

    call Algorithm A1 with $B = \lfloor (\max + \min)/2 \rfloor$

    if Algorithm A1 answers yes **then** min $= B$ **else** max $= B$

    return min // *opt* = min

    Observe that the invariant of the binary search is

    min $\leq$ optimal solution $<$ max

    Thus, if max $-$ min $= 1$, the optimal solution equals to min

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** (Cont'd)

    Since KANPSACK ∈ P, Algorithm A1 takes a time in $O(|\langle W, v_i, w_i, B\rangle|^c)$, for some $c > 0$

    Clearly, Algorithm A1 is called $O(\lg \sum v_i)$ times, each time with a different value for B.

    Since the value of B is always less than $\sum v_i$, Algorithm A2 takes a time in

$$O\left(\lg \sum v_i \cdot \left|\left\langle W, v_i, w_i, \sum v_i \right\rangle\right|^c\right)$$

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** (Cont'd)

    Since

$$\lg \sum v_i \leq \lg \sum (1 + v_i) \leq \lg \prod (1 + v_i)$$

$$= \sum \lg(1 + v_i) \leq |\langle W, v_i, w_i \rangle|$$

    We have

$$|\langle W, v_i, w_i, \sum v_i \rangle| = |\langle W, v_i, w_i \rangle| + |\langle \sum v_i \rangle|$$

$$= |\langle W, v_i, w_i \rangle| + \lg \sum v_i \leq 2 \cdot |\langle W, v_i, w_i \rangle|$$

Supplementary: NP-Hard

- Cook reducibility $\leq_T$
  - **THEOREM** (Cont'd)

    Hence, algorithm A2 takes a time in

$$O\left(\lg \sum v_i \cdot \left|\left\langle W, v_i, w_i, \sum v_i \right\rangle\right|^c\right)$$

$$= O(2^c |\langle W, v_i, w_i \rangle|^{c+1})$$

$$= O(|\langle W, v_i, w_i \rangle|^{c+1})$$

    which is a polynomial in terms of input size $|\langle W, v_i, w_i \rangle|$