COMS21400 : Time Complexity

G. Fuchsbauer

Dept of Computer Science
University of Bristol,
Room 3.53, Merchant Venturers Building

19–27 Nov 2012

Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness

BIG-O and small-o notation

Classify functions by their *asymptotic growth rate*

Let $f; g : \mathbb{N} \to \mathbb{R}^+$

- $f(n) = O(g(n))$ if

$$\exists c > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

("For some positive c: $f(n) \leq c \cdot g(n)$ for all sufficiently large n")

BIG-O and small-o notation

Classify functions by their *asymptotic growth rate*

Let $f; g \colon \mathbb{N} \to \mathbb{R}^+$

- $f(n) = O(g(n))$ if
$$\exists c > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$
("For some positive c: $f(n) \leq c \cdot g(n)$ for all sufficiently large n")

- $f(n) = o(g(n))$ if
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$
that is, $\forall c > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 : f(n) < c \cdot g(n)$
("For any positive c: $f(n) < c \cdot g(n)$ for all sufficiently large n")

Examples 1

Polynomials. If f is a polynomial of degree k then

$$f(n) = O(n^k)$$
$$f(n) = o(n^{k+1}), \text{ but } f \text{ is not } o(n^k)$$

- In general: if $0 < k_1 < k_2$ then $n^{k_1} = o(n^{k_2})$

Examples 1

Polynomials. If f is a polynomial of degree k then

$$f(n) = O(n^k)$$
$$f(n) = o(n^{k+1}), \text{ but } f \text{ is not } o(n^k)$$

- In general: if $0 < k_1 < k_2$ then $n^{k_1} = o(n^{k_2})$

Logarithms. $\left[\begin{array}{l} \text{Recall: } a^x = y \text{ then } x = \log_a y \\ \text{Typically } a = 2\text{: } (\lfloor \log_2 n \rfloor + 1) \text{ is } n\text{'s length in binary} \end{array}\right]$

- $\dfrac{(\log n)^k}{n^c} \xrightarrow[n\to\infty]{} 0 \quad \text{for all } k; c > 0$

thus $\log n = o(n^k)$, for all $k > 0$, $n \log n = o(n^2)$, ...

Examples 2

Exponentials. Any exponential function “dominates ” any polynomial:

- $\dfrac{n^k}{c^n} \underset{n\to\infty}{\longrightarrow} 0$ for all $k > 0; c > 1$

  thus $n^k = o(c^n)$, for any $c > 1$
- In general: if $c_1 < c_2$ then ${c_1}^n = o({c_2}^n)$

Notation: $f(n) = 2^{O(g(n))}$ iff $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq 2^{c \cdot g(n)}$

Outline

Time complexity for TMs

Definition. Let M be a TM which halts on every input. The **running time** or **time complexity** of M is $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on *any* input of length n.

("worst-case time")

Definition. Let $f : \mathbb{N} \to \mathbb{R}^+$. $\mathrm{TIME}(f(n))$ is the class of all languages decided by an $O(f(n))$-time TM.

Time complexity for TMs

Definition. Let M be a TM which halts on every input. The **running time** or **time complexity** of M is $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on *any* input of length n.

("worst-case time")

Definition. Let $f : \mathbb{N} \to \mathbb{R}^+$. $\mathrm{TIME}(f(n))$ is the class of all languages decided by an $O(f(n))$-time TM.

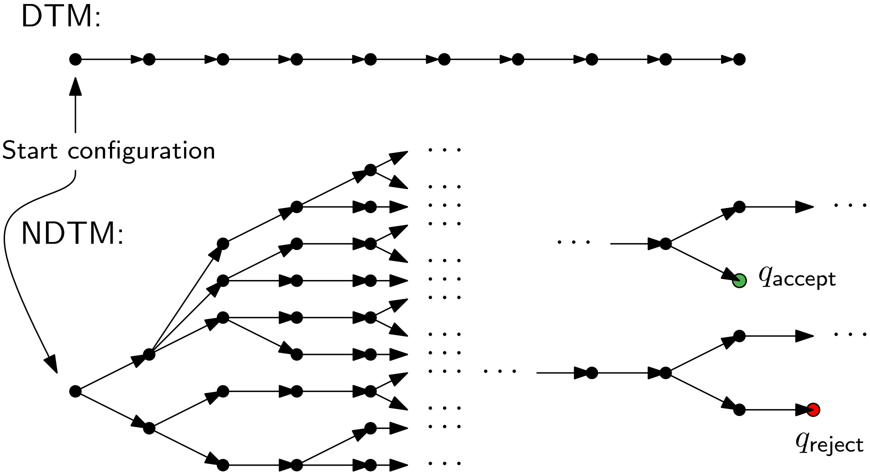
Examples.

- $\mathrm{TIME}(n)$ (linear time)
- $\mathrm{P} := \bigcup_k \mathrm{TIME}(n^k)$ (polynomial time)
- $\mathrm{EXP} := \bigcup_k \mathrm{TIME}(2^{n^k})$ (exponential time)

Gap: super-polynomial, sub-exponential ($\forall c > 1 : f(n) = o(c^n)$),
e.g.: $f(n) = n^{\log n}$.

Non-deterministic Turing machines

A node • is a configuration



Time complexity for NTMs

Definition. Let N be a NTM which is a decider. The **running time** of N is $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses *on any branch of its computation* on *any* input of length n.

Theorem. *(Time-complexity of NTM simulation.)* Every $t(n)$-time NTM N has an *equivalent* $2^{O(t(n))}$-time TM M.

(equivalent: N and M decide the same language.)

Outline

The class P

$$\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k)$$

- P is **robust** (not affected by model of computation)
- P is a mathematical **model** of “realistically solvable” or **“tractable”** problems (Cobham’s thesis)
(caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

The class P

$$P = \bigcup_k \mathrm{TIME}(n^k)$$

- P is **robust** (not affected by model of computation)
- P is a mathematical **model** of "realistically solvable" or **"tractable"** problems (Cobham's thesis)
  (caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

Examples

- *FACTORING* $\overset{?}{\in}$ P
  ($\mathit{FACTORING} = \{(N; M) \mid N$ has an integer factor $1 < k < M\}$)
  Brute force: $O(2^{n/2})$. Best known algo: $2^{O(n^{1/3}(\log n)^{2/3})}$

The class P

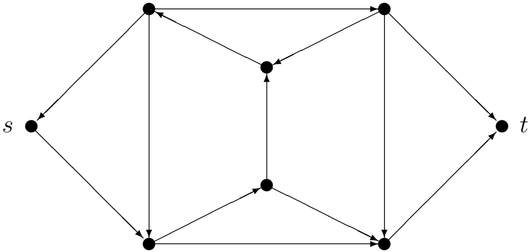
$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

- P is **robust** (not affected by model of computation)
- P is a mathematical **model** of “realistically solvable” or **“tractable”** problems (Cobham’s thesis)
  (caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

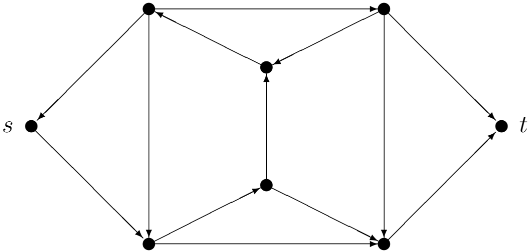
Examples

- $\mathit{FACTORING} \overset{?}{\in} \mathrm{P}$
  ($\mathit{FACTORING} = \{(N; M) \mid N$ has an integer factor $1 < k < M\}$)
  Brute force: $O(2^{n/2})$. Best known algo: $2^{O(n^{1/3}(\log n)^{2/3})}$
- $\mathit{PATH} \in \mathrm{P}$
  ($\mathit{PATH} = \{(G; s; t) \mid G$ is directed graph w/ path from s to $t\}$)

Examples

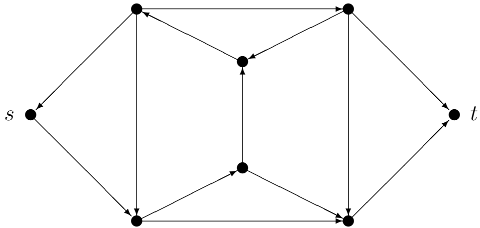


Examples



- $\mathit{HAMPATH} = \{(G; s; t) \mid G$ is directed graph with a Hamiltonian path from s to $t\}$

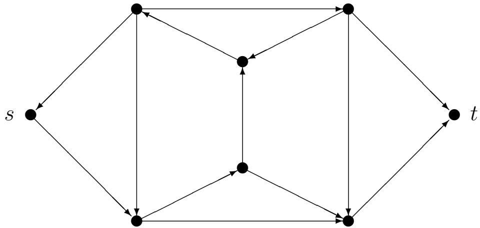
Examples



- $HAMPATH = \{(G; s; t) \mid G$ is directed graph with a Hamiltonian path from s to $t\}$
- Easy to check that $(G; s; t) \in HAMPATH$ when given a path;
  easy to check that $(N; M) \in FACTORING$ when given a factor

  ...

Examples



- *HAMPATH* $= \{(G; s; t) \mid G$ is directed graph with a Hamiltonian path from s to $t\}$
- Easy to check that $(G; s; t) \in$ *HAMPATH* when given a path;
  easy to check that $(N; M) \in$ *FACTORING* when given a factor
  …

Many computational problems

- can be solved by brute-force, testing exp. many *candidates*.
- Verification of desired property on a candidate is easy.

Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness

The class NP

Polynomial-time verifiers

- A **verifier** for a language A is a TM V, s.t.

$$A = \{w \mid V \text{ accepts } (w; c) \text{ for some } c\} \;.$$

The string c is called a **certificate** or **proof** of membership in A.

The class NP

Polynomial-time verifiers

- A **verifier** for a language A is a TM V, s.t.

$$A = \{w \mid V \text{ accepts } (w;c) \text{ for some } c\} \;:$$

  The string c is called a **certificate** or **proof** of membership in A.

- V is a **polynomial-time verifier** if runs in polynomial time in the length of w.

The class NP

Polynomial-time verifiers

- A **verifier** for a language A is a TM V, s.t.

$$A = \{w \mid V \text{ accepts } (w; c) \text{ for some } c\} \ :$$

  The string c is called a **certificate** or **proof** of membership in A.

- V is a **polynomial-time verifier** if runs in polynomial time in the length of w.

Definition. **NP** is the class of all **polynomially verifiable languages** i.e., all languages which have a polynomial-time verifier.

More on NP

$$\begin{aligned} \text{P} &= \text{class of languages that can be } \textbf{decided} \text{ ``quickly''} \\ \text{NP} &= \text{class of languages that can be } \textbf{verified} \text{ ``quickly''} \end{aligned}$$

- P $\subseteq$ NP (→ problems class)

More on NP

$$\begin{aligned} \text{P} &= \text{class of languages that can be } \textbf{decided} \text{ “quickly”} \\ \text{NP} &= \text{class of languages that can be } \textbf{verified} \text{ “quickly”} \end{aligned}$$

- P $\subseteq$ NP   ($\rightarrow$ problems class)

Definition. For a class of languages **C**, we define **co-C** as the class of all complements $\overline{A}$ of languages A in C.

- P $=$ co-P   ($\rightarrow$ problems class)
- NP $\stackrel{?}{=}$ co-NP

More on NP

$$\begin{aligned} \text{P} &= \text{class of languages that can be } \textbf{decided} \text{ “quickly”} \\ \text{NP} &= \text{class of languages that can be } \textbf{verified} \text{ “quickly”} \end{aligned}$$

- $\text{P} \subseteq \text{NP}$ $(\rightarrow$ problems class)

Definition. For a class of languages **C**, we define **co-C** as the class of all complements $\overline{A}$ of languages A in C.

- $\text{P} = \text{co-P}$ $(\rightarrow$ problems class)
- $\text{NP} \overset{?}{=} \text{co-NP}$

Examples. $\mathit{HAMPATH} \in \text{NP}$
$\mathit{COMPOSITES} = \{x \mid x = pq, \text{ for integers } p; q > 1\} \in \text{NP}$
$\mathit{PRIMES} \in \text{co-NP}$

Actually: $\mathit{PRIMES} \in \text{NP}$ (not obvious)
$\mathit{PRIMES} \in \text{P}$ (shown in 2003)

More on NP

$$\begin{aligned} \text{P} &= \text{class of languages that can be } \textbf{decided} \text{ "quickly"} \\ \text{NP} &= \text{class of languages that can be } \textbf{verified} \text{ "quickly"} \end{aligned}$$

- P $\subseteq$ NP   ($\rightarrow$ problems class)

Definition. For a class of languages **C**, we define **co-C** as the class of all complements $\overline{A}$ of languages A in C.

- P $=$ co-P   ($\rightarrow$ problems class)
- NP $\stackrel{?}{=}$ co-NP

Examples. *HAMPATH* $\in$ NP
COMPOSITES $= \{x \mid x = pq$, for integers $p, q > 1\} \in$ NP
PRIMES $\in$ co-NP

Actually: *PRIMES* $\in$ NP (not obvious)
PRIMES $\in$ P (shown in 2003)

$\overline{HAMPATH} \stackrel{?}{\in} NP$

The name NP

Theorem. A language is in NP iff it is decided by some polynomial-time NTM.

Corollary. NP $\subseteq$ EXP (by the theorem on Slide 9).

The name NP

Theorem. A language is in NP iff it is decided by some polynomial-time NTM.

Corollary. NP $\subseteq$ EXP (by the theorem on Slide 9).

$$\mathrm{P} \overset{?}{=} \mathrm{NP}$$

"Can every problem whose solution is quickly verifiable be solved quickly?"

- Implications?

The SATisfiability problem

- Boolean variables: x take values 1 (TRUE) or 0 (FALSE)
- Boolean operations: AND $(x_1 \wedge x_2)$, OR $(x_1 \vee x_2)$, NOT $(\overline{x})$
- Boolean formulas: e.g. $\quad = (\overline{x}_1 \wedge x_2) \ \vee \ ((x_1 \wedge \overline{x}_3) \vee x_2)$

The SATisfiability problem

- Boolean variables: x take values 1 (TRUE) or 0 (FALSE)
- Boolean operations: AND $(x_1 \wedge x_2)$, OR $(x_1 \vee x_2)$, NOT $(\overline{x})$
- Boolean formulas: e.g. $\phi = (\overline{x}_1 \wedge x_2) \vee ((x_1 \wedge \overline{x}_3) \vee x_2)$

A boolean formula is **satisfiable** if there exists an assignment of 0's and 1's to the variables, s.t. the formula evaluates to 1.

(Formula with n variables has 2^n possible assignments)

The SATisfiability problem

- Boolean variables: x take values 1 (TRUE) or 0 (FALSE)
- Boolean operations: AND $(x_1 \wedge x_2)$, OR $(x_1 \vee x_2)$, NOT $(\overline{x})$
- Boolean formulas: e.g. $\quad = (\overline{x}_1 \wedge x_2) \vee ((x_1 \wedge \overline{x}_3) \vee x_2)$

A boolean formula is **satisfiable** if there exists an assignment of 0's and 1's to the variables, s.t. the formula evaluates to 1.

(Formula with n variables has 2^n possible assignments)

$$\textbf{\textit{SAT}} := \{\langle \quad \rangle \mid \quad \text{is a satisfiable Boolean formula}\}$$

- $SAT \in$ NP, $\quad SAT$? co-NP

The SATisfiability problem

- Boolean variables: x take values 1 (TRUE) or 0 (FALSE)
- Boolean operations: AND $(x_1 \wedge x_2)$, OR $(x_1 \vee x_2)$, NOT $(\overline{x})$
- Boolean formulas: e.g. $\quad = (\overline{x}_1 \wedge x_2) \;\vee\; ((x_1 \wedge \overline{x}_3) \vee x_2)$

A boolean formula is **satisfiable** if there exists an assignment of 0's and 1's to the variables, s.t. the formula evaluates to 1.

(Formula with n variables has 2^n possible assignments)

$$\textbf{\textit{SAT}} := \{\langle \quad \rangle \mid \quad \text{is a satisfiable Boolean formula}\}$$

- $SAT \in$ NP, $\quad SAT \overset{?}{\in}$ co-NP

Theorem. *(Cook-Levin)*

$$SAT \in \mathrm{P} \;\text{ iff }\; \mathrm{P} = \mathrm{NP}$$

Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness

Reducibility

Informally: If A *reduces* to B then B is "harder" than A

(cf. undecidability)

Reducibility

Informally: If A *reduces* to B then B is "harder" than A

(cf. undecidability)

Definition. $f\colon \Sigma^* \to \Sigma^*$ is **polynomial-time computable** if there is a poly-time TM, which on input w halts with $f(w)$ on its tape.

Reducibility

Informally: If A *reduces* to B then B is "harder" than A

(cf. undecidability)

Definition. $f\colon \Sigma^* \to \Sigma^*$ is **polynomial-time computable** if there is a poly-time TM, which on input w halts with $f(w)$ on its tape.

Definition. A language A is **polynomial-time reducible** to B if there is a poly-time computable $f\colon \Sigma^* \to \Sigma^*$ with

$$w \in A \quad \text{iff} \quad f(w) \in B$$

We write $A \leq_p B$.

Reducibility

Informally: If A *reduces* to B then B is "harder" than A

(cf. undecidability)

Definition. $f\colon \Sigma^* \to \Sigma^*$ is **polynomial-time computable** if there is a poly-time TM, which on input w halts with $f(w)$ on its tape.

Definition. A language A is **polynomial-time reducible** to B if there is a poly-time computable $f\colon \Sigma^* \to \Sigma^*$ with

$$w \in A \quad \text{iff} \quad f(w) \in B$$

We write $A \leq_p B$.

Theorem. If $A \leq_p B$ and $B \in \mathrm{P}$ then $A \in \mathrm{P}$.

NP-completeness

Definition. A language B is **NP-complete** if

- $B \in$ NP, and
- every A in NP is polynomial-time reducible to B

(NP-complete problems are the "hardest" problems in NP)

NP-completeness

Definition. A language B is **NP-complete** if

- $B \in \mathrm{NP}$, and
- every A in NP is polynomial-time reducible to B

(NP-complete problems are the "hardest" problems in NP)

Theorem. If B is NP-complete and $B \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

NP-completeness

Definition. A language B is **NP-complete** if

- $B \in \mathrm{NP}$, and
- every A in NP is polynomial-time reducible to B

(NP-complete problems are the "hardest" problems in NP)

Theorem. If B is NP-complete and $B \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Theorem. If B is NP-complete and $B \leq_p C$, for $C \in \mathrm{NP}$, then C is NP-complete.

NP-completeness

Definition. A language B is **NP-complete** if

- $B \in \mathrm{NP}$, and
- every A in NP is polynomial-time reducible to B

(NP-complete problems are the “hardest” problems in NP)

Theorem. If B is NP-complete and $B \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Theorem. If B is NP-complete and $B \leq_p C$, for $C \in \mathrm{NP}$, then C is NP-complete.

Theorem. *(Cook-Levin, restated)* *SAT* is NP-complete.

3-SAT

- Literal: x or $\overline{x}$
- Clause: Disjunction of literals, e.g. $(x_1 \vee \overline{x}_2 \vee x_3)$
- is in conjunctive normal form if is a conjunction of clauses
- 3-CNF formula: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_2 \vee \overline{x}_5 \vee x_6) \wedge (x_3 \vee \overline{x}_6 \vee x_4)$.

3-SAT

- Literal: x or $\overline{x}$
- Clause: Disjunction of literals, e.g. $(x_1 \vee \overline{x}_2 \vee x_3)$
- is in conjunctive normal form if is a conjunction of clauses
- 3-CNF formula: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_2 \vee \overline{x}_5 \vee x_6) \wedge (x_3 \vee \overline{x}_6 \vee x_4)$.

$$\textit{3-SAT} := \{\langle \quad \rangle \mid \quad \text{is a satisfiable 3-CNF formula}\}$$

3-SAT

- Literal: x or $\overline{x}$
- Clause: Disjunction of literals, e.g. $(x_1 \vee \overline{x}_2 \vee x_3)$
- is in conjunctive normal form if is a conjunction of clauses
- 3-CNF formula: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_2 \vee \overline{x}_5 \vee x_6) \wedge (x_3 \vee \overline{x}_6 \vee x_4)$.

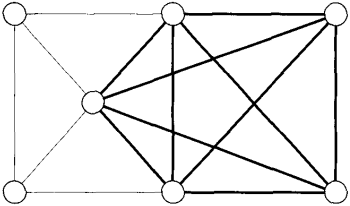
$$\textit{3-SAT} := \{\langle \quad \rangle \mid \quad \text{is a satisfiable 3-CNF formula}\}$$

Theorem. *3-SAT* is NP-complete.

More NP-complete languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G; k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

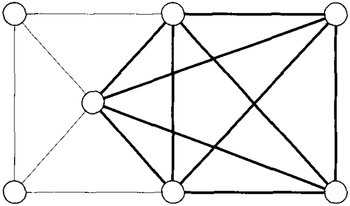


More NP-complete languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G;k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

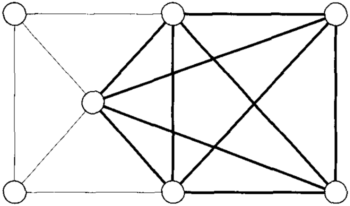
Theorem. *CLIQUE* is NP-complete.



More NP-complete languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G;k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$



Theorem. *CLIQUE* is NP-complete.

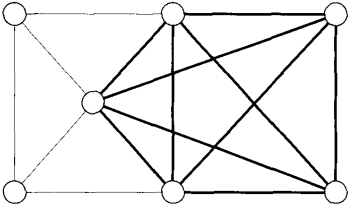
More NP-complete languages:

- *HAMPATH*

More NP-complete languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$



Theorem. *CLIQUE* is NP-complete.

More NP-complete languages:

- *HAMPATH*
- $SUBSET\text{-}SUM = \{\{x_1, \ldots, x_k\} \mid$ for some $\{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ we have: $\sum y_i = 0\}$