TECHNISCHE UNIVERSITÄT DRESDEN

FOUNDATIONS OF COMPLEXITY THEORY

Lecture 8: NP-Complete Problems

David Carral

Knowledge-Based Systems

TU Dresden, November 17, 2020

Towards More NP-Complete Problems

Starting with **Sat**, one can readily show more problems **P** to be NP-complete, each time performing two steps:

(1) Show that **P** $\in$ NP

(2) Find a known NP-complete problem **P′** and reduce **P′** $\leq_p$ **P**

Thousands of problems have now been shown to be NP-complete.
(See Garey and Johnson for an early survey)

In this course:

	$\leq_p$	**Clique**	$\leq_p$	**Independent Set**
Sat	$\leq_p$	**3-Sat**	$\leq_p$	**Dir. Hamiltonian Path**
	$\leq_p$	**Subset Sum**	$\leq_p$	**Knapsack**

3-Sat, Hamiltonian Path, and Subset Sum

NP-Completeness of **3-Sat**

3-Sat: Satisfiability of formulae in CNF with ≤ 3 literals per clause

Theorem 8.1: 3-Sat is NP-complete.

Proof: Hardness by reduction **Sat** $\leq_p$ **3-Sat**:

- Given: φ in CNF
- Construct φ' by replacing clauses $C_i = (L_1 \vee \cdots \vee L_k)$ with $k > 3$ by

$$C'_i := (L_1 \vee Y_1) \wedge (\neg Y_1 \vee L_2 \vee Y_2) \wedge \ldots \wedge (\neg Y_{k-1} \vee L_k)$$

  Here, the Y_j are fresh variables for each clause.
- Claim: φ is satisfiable iff φ' is satisfiable.

Example

Let $\varphi := (X_1 \vee X_2 \vee \neg X_3 \vee X_4) \quad \wedge \quad (\neg X_4 \vee \neg X_2 \vee X_5 \vee \neg X_1)$

Then $\varphi' :=$

$$
\begin{aligned}
&(X_1 \vee Y_1) \wedge \\
&(\neg Y_1 \vee X_2 \vee Y_2) \wedge \\
&(\neg Y_2 \vee \neg X_3 \vee Y_3) \wedge \\
&(\neg Y_3 \vee X_4) \wedge \\
&(\neg X_4 \vee Z_1) \wedge \\
&(\neg Z_1 \vee \neg X_2 \vee Z_2) \wedge \\
&(\neg Z_2 \vee X_5 \vee Z_3) \wedge \\
&(\neg Z_3 \vee \neg X_1)
\end{aligned}
$$

Proving NP-Completeness of 3-Sat

"⇒" Given $\varphi := \bigwedge_{i=1}^{m} C_i$ with clauses C_i, show that if φ is satisfiable then φ' is satisfiable

For a satisfying assignment β for φ, define an assignment β' for φ':

For each $C := (L_1 \vee \cdots \vee L_k)$, with $k > 3$, in φ there is

$$C' = (L_1 \vee Y_1) \ \wedge \ (\neg Y_1 \vee L_2 \vee Y_2) \ \wedge \ldots \ \wedge \ (\neg Y_{k-1} \vee L_k) \text{ in } \varphi'$$

As β satisfies φ, there is $i \leq k$ s.th. $\beta(L_i) = 1$ i.e. $\begin{array}{l} \beta(X) = 1 \text{ if } L_i = X \\ \beta(X) = 0 \text{ if } L_i = \neg X \end{array}$

$$\text{Set} \quad \begin{array}{ll} \beta'(Y_j) = 1 & \text{for } j < i \\ \beta'(Y_j) = 0 & \text{for } j \geq i \\ \beta'(X) = \beta(X) & \text{for all variables in } \varphi \end{array}$$

This is a satisfying asignment for φ'

Proving NP-Completeness of 3-SAT

"$\Leftarrow$" Show that if φ' is satisfiable then so is φ

Suppose β is a satisfying assignment for φ' – then β satisfies φ:

Let $C := (L_1 \vee \cdots \vee L_k)$ be a clause of φ

(1) If $k \leq 3$ then C is a clause of φ'

(2) If $k > 3$ then

$$C' = (L_1 \vee Y_1) \;\wedge\; (\neg Y_1 \vee L_2 \vee Y_2) \;\wedge \ldots \wedge\; (\neg Y_{k-1} \vee L_k) \text{ in } \varphi'$$

β must satisfy at least one L_i, $1 \leq i \leq k$

Case (2) follows since, if $\beta(L_i) = 0$ for all $i \leq k$ then C' can be reduced to

$$\begin{aligned} C' &= (Y_1) \;\wedge\; (\neg Y_1 \vee Y_2) \;\wedge \ldots \wedge\; (\neg Y_{k-1}) \\ &\equiv Y_1 \;\wedge\; (Y_1 \rightarrow Y_2) \;\wedge \ldots \wedge\; (Y_{k-2} \rightarrow Y_{k-1}) \;\wedge \neg Y_{k-1} \end{aligned}$$

which is not satisfiable. $\square$

NP-Completeness of Directed Hamiltonian Path

Directed Hamiltonian Path

Input: A directed graph G.

Problem: Is there a directed path in G containing every vertex exactly once?

Theorem 8.2: Directed Hamiltonian Path is NP-complete.

Proof:

(1) **Directed Hamiltonian Path** $\in$ NP:
Take the path to be the certificate.

(2) **Directed Hamiltonian Path** is NP-hard:
3-Sat $\leq_p$ **Directed Hamiltonian Path**

Digression: How to design reductions

Task: Show that problem **P** (**Directed Hamiltonian Path**) is NP-hard.

- Arguably, the most important part is to decide where to start from.

  That is, which problem to reduce to **Directed Hamiltonian Path**?

- Considerations:
  - Is there an NP-complete problem similar to **P**?
    (for example, **Clique** and **Independent Set**)
  - It is not always beneficial to choose a problem of the same type
    (for example, reducing a graph problem to a graph problem)
    - For instance, **Clique**, **Independent Set** are “local” problems
      (is there a set of vertices inducing some structure)
    - Hamiltonian Path is a global problem
      (find a structure – the Hamiltonian path – containing all vertices)

- How to design the reduction:
  - Does your problem come from an optimisation problem?
    If so: a maximisation problem? a minimisation problem?
  - Learn from examples, have good ideas.

NP-Completeness of Directed Hamiltonian Path

Directed Hamiltonian Path

Input: A directed graph G.

Problem: Is there a directed path in G containing every vertex exactly once?

Theorem 8.2: **Directed Hamiltonian Path** is NP-complete.

Proof:

(1) **Directed Hamiltonian Path** $\in$ NP:
Take the path to be the certificate.

(2) **Directed Hamiltonian Path** is NP-hard:
3-Sat $\leq_p$ **Directed Hamiltonian Path**

NP-Completeness of Directed Hamiltonian Path

Proof (Proof idea): (see blackboard for details)

Let $\varphi := \bigwedge_{i=1}^{k} C_i$ and $C_i := (L_{i,1} \vee L_{i,2} \vee L_{i,3})$

- For each variable X occurring in φ, we construct a directed graph ("gadget") that allows only two Hamiltonian paths: "true" and "false"
- Gadgets for each variable are "chained" in a directed fashion, so that all variables must be assigned one value
- Clauses are represented by vertices that are connected to the gadgets in such a way that they can only be visited on a Hamiltonian path that corresponds to an assignment where they are true

Details are also given in [Sipser, Theorem 7.46].

Example 8.3: $\varphi := C_1 \wedge C_2$ where $C_1 := (X \vee \neg Y \vee Z)$ and $C_2 := (\neg X \vee Y \vee \neg Z)$ (see blackboard)

Towards More NP-Complete Problems

Starting with **Sat**, one can readily show more problems **P** to be NP-complete, each time performing two steps:

(1) Show that **P** $\in$ NP

(2) Find a known NP-complete problem **P′** and reduce **P′** $\leq_p$ **P**

Thousands of problems have now been shown to be NP-complete.
(See Garey and Johnson for an early survey)

In this course:

	$\leq_p$	**Clique**	$\leq_p$	**Independent Set**
Sat	$\leq_p$	**3-Sat**	$\leq_p$	**Dir. Hamiltonian Path**
	$\leq_p$	**Subset Sum**	$\leq_p$	**Knapsack**

NP-Completeness of Subset Sum

Subset Sum

Input: A collection[1] of positive integers $S = \{a_1, \ldots, a_k\}$ and a target integer t.

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

Theorem 8.4: **Subset Sum** is NP-complete.

Proof:

(1) **Subset Sum** $\in$ NP: Take T to be the certificate.

(2) **Subset Sum** is NP-hard: **Sat** $\leq_p$ **Subset Sum**

[1]) This "collection" is supposed to be a multi-set, i.e., we allow the same number to occur several times. The solution "subset" can likewise use numbers multiple times, but not more often than they occured in the given collection.

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

$$\begin{array}{lc|ccccc|ccc}
 & & X_1 & X_2 & X_3 & X_4 & X_5 & C_1 & C_2 & C_3 \\
t_1 & = & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_1 & = & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
t_2 & = & & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_2 & = & & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
t_3 & = & & & 1 & 0 & 0 & 1 & 0 & 0 \\
f_3 & = & & & 1 & 0 & 0 & 0 & 0 & 1 \\
t_4 & = & & & & 1 & 0 & 0 & 0 & 1 \\
f_4 & = & & & & 1 & 0 & 0 & 1 & 0 \\
t_5 & = & & & & & 1 & 0 & 0 & 1 \\
f_5 & = & & & & & 1 & 0 & 0 & 0 \\
m_{1,1} & = & & & & & & 1 & 0 & 0 \\
m_{1,2} & = & & & & & & 1 & 0 & 0 \\
m_{2,1} & = & & & & & & 0 & 1 & 0 \\
m_{3,1} & = & & & & & & 0 & 0 & 1 \\
m_{3,2} & = & & & & & & 0 & 0 & 1 \\
m_{3,3} & = & & & & & & 0 & 0 & 1 \\
\hline
t & = & 1 & 1 & 1 & 1 & 1 & 3 & 2 & 4
\end{array}$$

Sat $\leq_p$ Subset Sum

Given: $\varphi := C_1 \wedge \cdots \wedge C_k$ in conjunctive normal form.

(w.l.o.g. at most 9 literals per clause)

Let $X_1, \ldots, X_n$ be the variables in φ. For each X_i let

$$t_i := a_1 \ldots a_n c_1 \ldots c_k \text{ where } a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \text{ and } c_j := \begin{cases} 1 & X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}$$

$$f_i := a_1 \ldots a_n c_1 \ldots c_k \text{ where } a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \text{ and } c_j := \begin{cases} 1 & \neg X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}$$

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		X_1	X_2	X_3	X_4	X_5	C_1	C_2	C_3
t_1	=	1	0	0	0	0	1	0	0
f_1	=	1	0	0	0	0	0	1	0
t_2	=		1	0	0	0	1	0	0
f_2	=		1	0	0	0	0	0	1
t_3	=			1	0	0	1	0	0
f_3	=			1	0	0	0	0	1
t_4	=				1	0	0	0	1
f_4	=				1	0	0	1	0
t_5	=					1	0	0	1
f_5	=					1	0	0	0
$m_{1,1}$	=						1	0	0
$m_{1,2}$	=						1	0	0
$m_{2,1}$	=						0	1	0
$m_{3,1}$	=						0	0	1
$m_{3,2}$	=						0	0	1
$m_{3,3}$	=						0	0	1
t	=	1	1	1	1	1	3	2	4

Sat $\leq_p$ Subset Sum

Further, for each clause C_i take $r := |C_i| - 1$ integers $m_{i,1}, \ldots, m_{i,r}$

where $m_{i,j} := c_i \ldots c_k$ with $c_\ell := \begin{cases} 1 & \ell = i \\ 0 & \ell \neq i \end{cases}$

Definition of S: Let

$$S := \{t_i, f_i \mid 1 \leq i \leq n\} \cup \{m_{i,j} \mid 1 \leq i \leq k, \quad 1 \leq j \leq |C_i| - 1\}$$

Target: Finally, choose as target

$$t := a_1 \ldots a_n c_1 \ldots c_k \text{ where } a_i := 1 \text{ and } c_i := |C_i|$$

Claim: There is $T \subseteq S$ with $\sum_{a_i \in T} a_i = t$ iff φ is satisfiable.

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		X_1	X_2	X_3	X_4	X_5	C_1	C_2	C_3
t_1	=	1	0	0	0	0	1	0	0
f_1	=	1	0	0	0	0	0	1	0
t_2	=		1	0	0	0	1	0	0
f_2	=		1	0	0	0	0	0	1
t_3	=			1	0	0	1	0	0
f_3	=			1	0	0	0	0	1
t_4	=				1	0	0	0	1
f_4	=				1	0	0	1	0
t_5	=					1	0	0	1
f_5	=					1	0	0	0
$m_{1,1}$	=						1	0	0
$m_{1,2}$	=						1	0	0
$m_{2,1}$	=						0	1	0
$m_{3,1}$	=						0	0	1
$m_{3,2}$	=						0	0	1
$m_{3,3}$	=						0	0	1
t	=	1	1	1	1	1	3	2	4

NP-Completeness of Subset Sum

Let $\varphi := \bigwedge C_i$ $\qquad$ C_i: clauses

Show: If φ is satisfiable, then there is $T \subseteq S$ with $\sum_{s \in T} s = t$.

Let β be a satisfying assigment for φ

Set $$T_1 := \{t_i \mid \beta(X_i) = 1,\ 1 \leq i \leq m\} \cup \{f_i \mid \beta(X_i) = 0,\ 1 \leq i \leq m\}$$

Further, for each clause C_i let r_i be the number of satisfied literals in C_i (with resp. to β).

Set $T_2 := \{m_{i,j} \mid 1 \leq i \leq k, \quad 1 \leq j \leq |C_i| - r_i\}$

and define $T := T_1 \cup T_2$.

It follows: $\sum_{s \in T} s = t$

NP-Completeness of **Subset Sum**

Show: If there is $T \subseteq S$ with $\sum_{s \in T} s = t$, then φ is satisfiable.

Let $T \subseteq S$ such that $\sum_{s \in T} s = t$

Define $\beta(X_i) = \begin{cases} 1 & \text{if } t_i \in T \\ 0 & \text{if } f_i \in T \end{cases}$

This is well defined as for all i: $t_i \in T$ or $f_i \in T$ but not both.

Further, for each clause, there must be one literal set to 1 as for all i,

the $m_{i,j} \in S$ do not sum up to the number of literals in the clause. $\square$

Towards More NP-Complete Problems

Starting with **Sat**, one can readily show more problems **P** to be NP-complete, each time performing two steps:

(1) Show that **P** $\in$ NP

(2) Find a known NP-complete problem **P′** and reduce **P′** $\leq_p$ **P**

Thousands of problems have now been shown to be NP-complete.
(See Garey and Johnson for an early survey)

In this course:

	$\leq_p$	**Clique**	$\leq_p$	**Independent Set**
Sat	$\leq_p$	**3-Sat**	$\leq_p$	**Dir. Hamiltonian Path**
	$\leq_p$	**Subset Sum**	$\leq_p$	**Knapsack**

NP-completeness of KNAPSACK

KNAPSACK

Input: A set $I := \{1, \ldots, n\}$ of items
each of value v_i and weight w_i for $1 \leq i \leq n$,
target value t and weight limit ℓ

Problem: Is there $T \subseteq I$ such that
$\sum_{i \in T} v_i \geq t$ and $\sum_{i \in T} w_i \leq \ell$?

Theorem 8.5: KNAPSACK is NP-complete.

Proof:

(1) **KNAPSACK** $\in$ NP: Take T to be the certificate.

(2) **KNAPSACK** is NP-hard: **SUBSET SUM** $\leq_p$ **KNAPSACK**

Subset Sum $\leq_p$ Knapsack

	Given:	$S := \{a_1, \ldots, a_n\}$	collection of positive integers
Subset Sum:		t	target integer
	Problem:	Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?	

Reduction: From this input to **Subset Sum** construct

- set of items $I := \{1, \ldots, n\}$
- weights and values $v_i = w_i = a_i$ for all $1 \leq i \leq n$
- target value $t' := t$ and weight limit $\ell := t$

Clearly: For every $T \subseteq S$

$$\sum_{a_i \in T} a_i = t \qquad \text{iff} \qquad \begin{aligned} \textstyle\sum_{a_i \in T} v_i \geq t' &= t \\ \textstyle\sum_{a_i \in T} w_i \leq \ell &= t \end{aligned}$$

Hence: The reduction is correct and in polynomial time.

A Polynomial Time Algorithm for Knapsack

Knapsack can be solved in time $O(n\ell)$ using dynamic programming

Initialisation:

- Create an $(\ell + 1) \times (n + 1)$ matrix M
- Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$

Computation: Assign further $M(w, i)$ to be the largest total value obtainable by selecting from the first i items with weight limit w:

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1)$ as

$$M(w, i + 1) := \max \{M(w, i),\ M(w - w_{i+1}, i) + v_{i+1}\}$$

Here, if $w - w_{i+1} < 0$ we always take $M(w, i)$.

Acceptance: If M contains an entry $\geq t$, accept. Otherwise reject.

Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1$ $v_2 = 3$ $v_3 = 4$ $v_4 = 2$

Weight: $w_1 = 1$ $w_2 = 1$ $w_3 = 3$ $w_4 = 2$

Weight limit: $\ell = 5$ Target value: $t = 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3	3	3
2	0	1	4	4	4
3	0	1	4	4	5
4	0	1	4	7	7
5	0	1	4	8	8

Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$ For $i = 0, 1, \ldots, n-1$ set $M(w, i+1) := \max\{M(w, i),\ M(w - w_{i+1}, i) + v_{i+1}\}$

A Polynomial Time Algorithm for Knapsack

Knapsack can be solved in time $O(n\ell)$ using dynamic programming

Initialisation:

- Create an $(\ell + 1) \times (n + 1)$ matrix M
- Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$

Computation: Assign further $M(w, i)$ to be the largest total value obtainable by selecting from the first i items with weight limit w:

For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1)$ as

$$M(w, i + 1) := \max \{M(w, i),\ M(w - w_{i+1}, i) + v_{i+1}\}$$

Here, if $w - w_{i+1} < 0$ we always take $M(w, i)$.

Acceptance: If M contains an entry $\geq t$, accept. Otherwise reject.

Example

Input $I = \{1, 2, 3, 4\}$ with

Values: $v_1 = 1 \quad v_2 = 3 \quad v_3 = 4 \quad v_4 = 2$

Weight: $w_1 = 1 \quad w_2 = 1 \quad w_3 = 3 \quad w_4 = 2$

Weight limit: $\ell = 5$ Target value: $t = 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3	3	3
2	0	1	4	4	4
3	0	1	4	4	5
4	0	1	4	7	7
5	0	1	4	8	8

Set $M(w, 0) := 0$ for all $1 \leq w \leq \ell$ and $M(0, i) := 0$ for all $1 \leq i \leq n$ For $i = 0, 1, \ldots, n - 1$ set $M(w, i + 1) := \max\{M(w, i),\ M(w - w_{i+1}, i) + v_{i+1}\}$

Did we prove P $=$ NP?

Summary:

- Theorem 8.5: **Knapsack** is NP-complete
- **Knapsack** can be solved in time $O(n\ell)$ using dynamic programming

What went wrong?

> **Knapsack**
>
> Input: A set $I := \{1, \ldots, n\}$ of items
> each of value v_i and weight w_i for $1 \leq i \leq n$,
> target value t and weight limit ℓ
>
> Problem: Is there $T \subseteq I$ such that
> $\sum_{i \in T} v_i \geq t$ and $\sum_{i \in T} w_i \leq \ell$?

Pseudo-Polynomial Time

The previous algorithm is not sufficient to show that **Knapsack** is in P

- The algorithm fills a $(\ell + 1) \times (n + 1)$ matrix M
- The size of the input to **Knapsack** is $O(n \log \ell)$

$\rightsquigarrow$ the size of M is not bounded by a polynomial in the length of the input!

Definition 8.6 (Pseudo-Polynomial Time): Problems decidable in time polynomial in the sum of the input length and the value of numbers occurring in the input.

Equivalently: Problems decidable in polynomial time when using unary encoding for all numbers in the input.

- If **Knapsack** is restricted to instances with $\ell \leq p(n)$ for a polynomial p, then we obtain a problem in P.
- **Knapsack** is in polynomial time for unary encoding of numbers.

Strong NP-completeness

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

Examples:

- **Knapsack**
- **Subset Sum**

Strong NP-completeness: Problems which remain NP-complete even if all numbers are bounded by a polynomial in the input length (equivalently: even for unary coding of numbers).

Examples:

- **Clique**
- **Sat**
- **Hamiltonian Cycle**
- ...

Note: Showing **Sat** $\leq_p$ **Subset Sum** required exponentially large numbers.

Beyond NP

The Class coNP

Recall that coNP is the complement class of NP.

Definition 8.7:

- For a language $\mathbf{L} \subseteq \Sigma^*$ let $\overline{\mathbf{L}} := \Sigma^* \setminus \mathbf{L}$ be its complement
- For a complexity class C, we define $\text{coC} := \{\mathbf{L} \mid \overline{\mathbf{L}} \in \text{C}\}$
- In particular $\text{coNP} = \{\mathbf{L} \mid \overline{\mathbf{L}} \in \text{NP}\}$

A problem belongs to coNP, if no-instances have short certificates.

Examples:

- **No Hamiltonian Path**: Does the graph G not have a Hamiltonian path?
- **Tautology**: Is the propositional logic formula φ a tautology (true under all assignments)?
- …

coNP-completeness

Definition 8.8: A language $\mathbf{C} \in$ coNP is coNP-complete, if $\mathbf{L} \leq_p \mathbf{C}$ for all $\mathbf{L} \in$ coNP.

Theorem 8.9:

(1) P = coP

(2) Hence, P $\subseteq$ NP $\cap$ coNP

Open questions:

- NP = coNP?

  Most people do not think so.

- P = NP $\cap$ coNP?

  Again, most people do not think so.

Summary and Outlook

3-Sat and **Hamiltonian Path** are also NP-complete

So are **SubSet Sum** and **Knapsack**, but only if numbers are encoded effiently (pseudo-polynomial time)

There do not seem to be polynomial certificates for coNP instances; and for some problems there seem to be certificates neither for instances nor for non-instances

What's next?

- Space
- Games
- Relating complexity classes