Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 15: Counting Classes, PSPACE and Beyond

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Counting problems
  - Examples
  - The class #**P**
  - Reductions and completeness
- Polynomial space
  - The QSAT problem
  - Two-person games, games against nature, interactive protocols
- A glimpse beyond

(C. Papadimitriou: *Computational Complexity*, Chapters 18–20)

Counting Problems

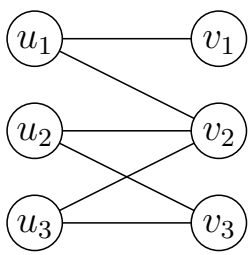
- At this lecture we shall briefly consider *counting problems* associated to **NP**-type decision problems, asking not just if a certificate (solution) to a problem instance exists, but *how many* there are.
- #SAT: given a Boolean expression, compute the number of different truth assignments that satisfy it.
- #HAMILTON PATH: compute the number of different Hamilton paths in a given graph.

Case I: #MATCHING aka PERMANENT

- Counting the number of solutions can be highly nontrivial even if the corresponding decision problem is solvable in $\mathbf{P}$.
- An example is the problem of counting the number of perfect matchings of a bipartite graph.
- This corresponds to the problem of computing the *permanent* of a matrix

$$\operatorname{perm} A^G = \sum_{\pi} \prod_{i=1}^{n} A^G_{i,\pi(i)} = \sum_{\pi} A^G_{1,\pi(1)} A^G_{2,\pi(2)} \cdots A^G_{n,\pi(n)}$$

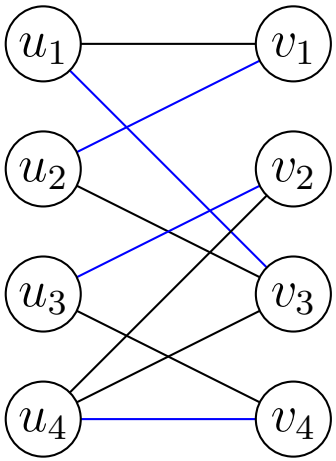
  where A^G is the adjacency matrix of the graph.
- This is why the problem is often called PERMANENT.



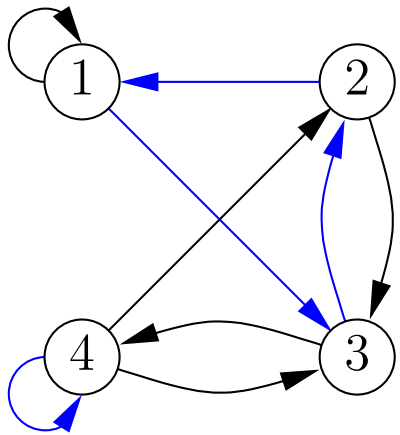
$$A^G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{aligned} \operatorname{perm} A^G &= A^G_{1,1}A^G_{2,2}A^G_{3,3} + A^G_{1,1}A^G_{2,3}A^G_{3,2} + \\ &= A^G_{1,2}A^G_{2,1}A^G_{3,3} + A^G_{1,2}A^G_{2,3}A^G_{3,1} + \\ &= A^G_{1,3}A^G_{2,1}A^G_{3,2} + A^G_{1,3}A^G_{2,2}A^G_{3,1} \end{aligned}$$

- A bipartite graph G with parts $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$ can be seen as a directed graph G' with vertices $\{1, \ldots, n\}$ where (i,j) is an edge in G' iff $\{u_i, v_j\}$ is an edge in G.
- Now a perfect matching corresponds to a *cycle cover*: a set of vertex-disjoint cycles that together cover all the vertices.



$$A^G = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$



For instance, a perfect matching $\{\{u_1, v_3\}, \{u_3, v_2\}, \{u_2, v_1\}, \{u_4, v_4\}\}$ corresponds to the cycle cover $\{(1\ 3\ 2), (4)\}$.

Case II: Graph Reliability

- Counting solutions is relevant, e.g., to probabilistic calculations.
- GRAPH RELIABILITY: count the number of subgraphs of a graph that contain a path from 1 to n.
  This number (divided by the number of subgraphs) gives the reliability of the graph: the probability that two vertices remain connected if all edges fail independently with probability $\frac{1}{2}$.

The class #P

- Let Q be a polynomially balanced and polynomial-time decidable binary relation.
- The *counting problem* associated with Q is the following:

  Given x, how many y's are there such that $(x, y) \in Q$?

  The answer is given as a binary integer.
- The class #**P** is the class of all counting problems associated with polynomially balanced and polynomial-time decidable binary relations.
- For #SAT, the relation Q is:
  $(x, y) \in Q$ iff the truth assignment y satisfies the Boolean expression x.
- For #HAMILTON PATH, the relation Q is:
  $(x, y) \in Q$ iff y is a Hamilton path of the graph x.

#P-Completeness

- Counting problems can be ordered using function problem reductions (R, S):
  - R for transforming an instance x of A to an instance $R(x)$ of B, and
  - S for transforming the number y of solutions of $R(x)$ to the number $S(y)$ of solutions of x
- Many such reductions are in fact *parsimonious*.
- A parsimonious reduction from a counting problem A to a counting problem B is a function R which maps an instance x of A to an instance $R(x)$ of B such that the number of solutions of $R(x)$ (for the problem B) is the same as that of x (for the problem A).
- Most reductions between **NP**-complete problems presented previously are parsimonious.
- A counting problem in #**P** is #**P**-*complete* if every problem in #**P** can be reduced to it.

Theorem

#SAT is #$\mathbf{P}$-complete

Proof

We have already argued that #SAT $\in$ #$\mathbf{P}$ holds.

Given $A \in$ #$\mathbf{P}$ with relation Q there is a poly-time TM M deciding Q. We can build a circuit $C(x)$ with $|x|^k$ inputs such that with input y the output of $C(x)$ is true iff M accepts $x; y$ (Cook's theorem).

This is a parsimonious reduction from A to #CIRCUIT SAT.

We have introduced a reduction from CIRCUIT SAT to SAT. It can be shown that this reduction is also parsimonious: the number of truth assignments where the output of the circuit is true coincides with the number of satisfying truth assignments of the corresponding set of clauses.

From this it follows that A reduces to #SAT parsimoniously as parsimonious reductions compose.

- Most of the reductions for decision problems that we have considered are parsimonious and this implies directly that many counting versions of **NP**-complete problems are #**P**-complete.
- #HAMILTON PATH is #**P**-complete.
- A polynomial algorithm for a search problem *does not* imply that the respective counting problem is solvable in polynomial time.
- Classical examples are PERMANENT and 2SAT.
- The corresponding search problems (perfect matching in a bipartite graph, satisfying truth assignment for 2cnf formula) are solvable in polynomial time.

Theorem (L. Valiant 1979)

*PERMANENT is #**P**-complete.*

Theorem

*#2SAT is #**P**-complete.*

The class #P—cont'd

- Notice that #$\mathbf{P}$ problems can be solved in polynomial space.
- How do $\mathbf{PH}$ and #$\mathbf{P}$ relate?
  (Remember: $\mathbf{PH} \subseteq \mathbf{PSPACE}$).
- Counting is stronger than the polynomial hierarchy!
- *Toda's theorem*: $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$.
- In fact $\mathbf{P}^{\#\mathbf{P}} = \mathbf{P}^{\mathbf{PP}}$ (and hence $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}$), where $\mathbf{PP}$ effectively tells only whether the *first bit* of the number of accepting computations is zero or one.

Polynomial Space

- The complexity class **PSPACE** emerges ubiquitously in many settings where there are two (or more) interacting agents.
- Examples include two-player games, decision-making under uncertainty ("games against nature"), interactive proof systems, verification of distributed systems,...
- The underlying reason for this is the characterisation of **PSPACE** in terms of alternating polynomial time computations.

The QSAT Problem

- QSAT (QBF): given a Boolean expression ϕ in CNF with variables $x_1, \ldots, x_n$, is there a truth value for variable x_1 such that for both truth values of x_2 there is a truth value for x_3 and so on up to x_n, such that ϕ is satisfied by the overall truth assignment?

$$\exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \phi$$

- Example:
  $\exists x_1 \forall x_2 \exists x_3 \forall x_4 : (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$
- QSAT is a generalisation of the Σ_i^p-complete problems QSAT_i.

QSAT, Alternation and PSPACE

- Recall that $\mathbf{AP} = \mathbf{ATIME}(n^k)$ is the class of languages decided in polynomial time by *alternating* Turing machines.
- It is known that $\mathbf{AP} = \mathbf{PSPACE}$. (The proof is basically an extension of Savitch's Theorem showing that $\mathbf{NPSPACE} = \mathbf{PSPACE}$.)
- Similarly as the problems QSAT_i are complete for the classes $\Sigma_i^p = \Sigma_i\mathbf{P}$, it is relatively easy to show that QSAT is complete for the class $\mathbf{AP}$, and hence $\mathbf{PSPACE}$.

Games

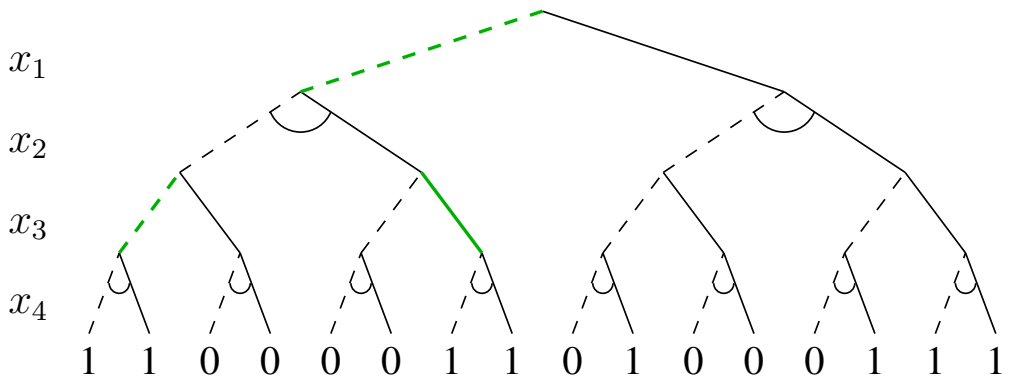
- QSAT is an example of a *two-person game*:
  - two players: $\exists$ and $\forall$
  - players move alternatingly ($\exists$ first)
  - a move: choosing the truth value of a variable
  - $\exists$ tries to make the formula ϕ true and $\forall$ false.
  - after n moves, either $\exists$ or $\forall$ wins.
- Example:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 : (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

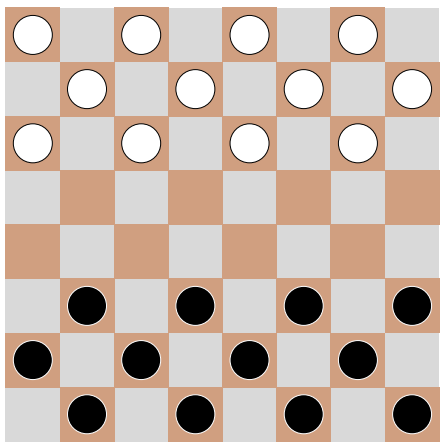
Games

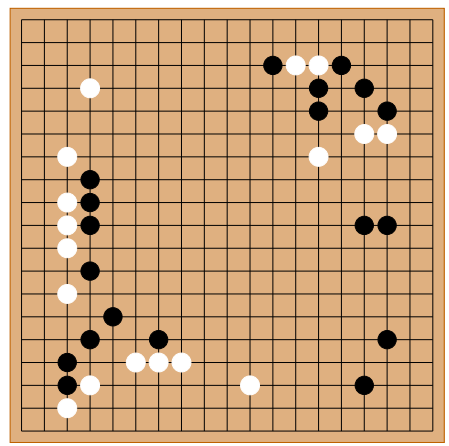
- QSAT is an example of a *two-person game*:
  - two players: $\exists$ and $\forall$
  - players move alternatingly ($\exists$ first)
  - a move: choosing the truth value of a variable
  - $\exists$ tries to make the formula ϕ true and $\forall$ false.
  - after n moves, either $\exists$ or $\forall$ wins.
- Example:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 : (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$



- A board game
  - two players move alternatingly on a “board”.
  - the number of moves is bounded by a polynomial in the size of the board
  - In the end some positions are considered a win of one player and the rest of the other.
- Solution: a winning strategy (typically an exponential object).
- Examples: chess, checkers, Go, nim, tic-tac-toe . . .





- How to separate computationally hard games from easy ones?
- Complexity theory cannot be used directly because games are played typically on a *fixed size* board.
- A possible solution: generalise the game to an arbitrary size board.
- GEOGRAPHY game:
  - Two players: I and II and board G (directed graph).
  - Move: select an unvisited neighbour of the current vertex.
  - The first player that cannot continue loses.
- GEOGRAPHY: Given a graph G and a starting vertex v, does player I have a winning strategy?
- GEOGRAPHY is **PSPACE**-complete.
- GO is **PSPACE**-complete.

Games Against Nature

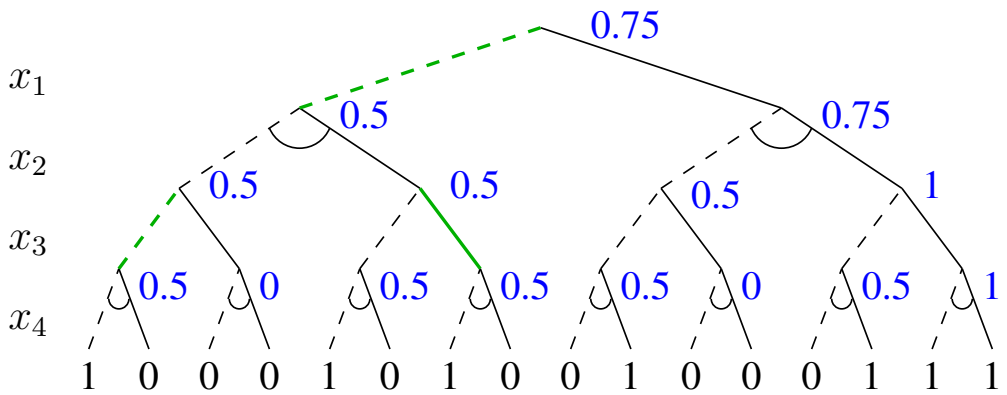
- More properly: decision making under uncertainty.
- The task is to make repeatedly a decision followed by a random event and we wish to design a strategy that optimises the outcome.
- Can be viewed as a game against “nature” which plays at random.
- Finding a strategy that maximises the probability of winning turns out to be computationally as hard as playing against an optimising opponent.

- There is a version of satisfiability and a version of alternating Turing machines that capture decision making under uncertainty.
- SSAT (Stochastic satisfiability): given a Boolean expression ϕ, is there a truth value for x_1 such that if the truth value of x_2 is *selected at random*, there is a truth value for x_3 and so on, such that the probability that ϕ is finally satisfied is greater than $\frac{1}{2}$?

$$\exists x_1 \mathrm{R} x_2 \exists x_3 \mathrm{R} x_4 \cdots \mathbf{prob}[\phi(x_1, \ldots, x_n) = \mathbf{true}] > 1/2$$

- Example:

$$\exists x_1 \mathrm{R} x_2 \exists x_3 \mathrm{R} x_4 : (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$$



- A probabilistic alternating polynomial-time Turing machine:
  - a precise nondeterministic machine with uniformly two nondeterministic choices alternating between states in two disjoint sets K_+ and K_{MAX}.
  - M accepts x if for each K_{MAX} state there is a choice of one of the two successors such that if we consider the resulting computation tree, a majority of the leaves is accepting.
- **APP**: languages decidable by APP machines.
- $\mathbf{APP} = \mathbf{PSPACE}$
- SSAT is **PSPACE**-complete

Interactive Protocols

- Interactive proof systems are closely related to *bounded* probabilistic alternating polynomial-time Turing machines (**ABPP**).
- ABPP machine:
  - if $x \in L$, then for each K_{MAX} state *there is* a choice of one of the two successors such that if we consider the resulting computation tree, 3/4 of the leaves are accepting in the resulting tree and
  - if $x \notin L$, then *for all* choices of successors in K_{MAX} states at most 1/4 of the leaves are accepting in the resulting tree.
- $\mathbf{ABPP} \subseteq \mathbf{IP} \subseteq \mathbf{APP}$
- $\mathbf{IP} = \mathbf{PSPACE}$ (A. Shamir 1992)
- $\mathbf{ABPP} = \mathbf{APP} = \mathbf{IP} = \mathbf{PSPACE}$

Learning Objectives on Counting and PSPACE

- The concept of counting problems
- The different facets of **PSPACE**: alternation, games, interactive proofs,. . .
- The classes #**P** and **PSPACE**
- Parsimonious reductions and completeness for counting problems
- Typical complete problems for #**P** and **PSPACE**
- The relationships of #**P** and **PSPACE** to other complexity classes

A Glimpse Beyond PSPACE

- $\mathbf{EXP} = \mathbf{TIME}(2^{n^k})$
- $\mathbf{NEXP} = \mathbf{NTIME}(2^{n^k})$
- If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{EXP} = \mathbf{NEXP}$.
- If $f(n)$ and $g(n) \geq n$ are proper complexity functions, then $\mathbf{TIME}(f(n)) = \mathbf{NTIME}(f(n))$ implies $\mathbf{TIME}(g(f(n))) = \mathbf{NTIME}(g(f(n)))$.
- $\mathbf{EXP} = \mathbf{APSPACE}$
- $\mathbf{EXP}/\mathbf{NEXP}$ provably intractable (not included in $\mathbf{P}$)!

Succinct Problems

- A type of **EXP**/**NEXP**-complete problems: **EXP** and **NEXP** are **P** and **NP** but on *exponentially more succinct* input.
- A succinct representation of a graph with n vertices where $n = 2^b$ as a Boolean circuit C:
  C has $2b$ input gates and its value on two b bit integers i, j is **true** iff there is a edge $[i, j]$ in the graph.
- SUCCINCT HAMILTON PATH: Given a succinct representation C of a graph G_C, does G_C have a Hamilton path?
- SUCCINCT CIRCUIT SAT, SUCCINCT 3SAT and SUCCINCT HAMILTON PATH are **NEXP**-complete.
- SUCCINCT CIRCUIT VALUE is **EXP**-complete.

First-Order Satisfiability

- The general case undecidable
- Some decidable cases of first-order logic (no function symbols or equality):
  - Schönfinkel-Bernays class: $\exists x_1 \ldots \exists x_k \forall y_1 \ldots \forall y_l \phi$ (**NEXP**-complete)
  - Ackermann class: $\exists x_1 \ldots \exists x_k \forall y \exists x_{k+1} \ldots \exists x_l \phi$ (**EXP**-complete)
  - Gödel class: $\exists x_1 \ldots \exists x_k \forall y_1 \forall y_2 \exists x_{k+1} \ldots \exists x_l \phi$ (**NEXP**-complete)
  - Monadic class (**NEXP**-complete)

Beyond NEXP

- $\mathbf{EXPSPACE} = \mathbf{SPACE}(2^{n^k})$
- $\text{2-}\mathbf{EXP} = \mathbf{TIME}(2^{2^{n^k}})$

  $\text{2-}\mathbf{NEXP} = \mathbf{NTIME}(2^{2^{n^k}})$
- $\text{3-}\mathbf{EXP} = \mathbf{TIME}(2^{2^{2^{n^k}}})$

  . . . [Provably true hierarchy here]
- $\mathbf{ELEMENTARY} = \mathbf{TIME}(2^{2^{2^{\vdots^{2^n}}}})$
- **REC**: recursive languages
- **NEXP/EXP**-complete problems are provably not in **P**.
- There are natural problems even in the higher classes:
  - Regular expression equivalence over $\{\cdot, *, \cup\}$ is **PSPACE**-complete.
  - Regular expression equivalence over $\{\cdot, *, \cup, \cap\}$ is **EXPSPACE**-complete.
  - Regular expression equivalence over $\{\cdot, *, \cup, \neg\}$ is not even in **ELEMENTARY**.

One Slide Summary: Computational Complexity Theory

- $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP} \subseteq \mathbf{EXPSPACE} \subseteq \mathbf{2\text{-}EXP} \subseteq \mathbf{ELEMENTARY} \subseteq \mathbf{REG}$
- Reductions:

  A reduces to $B \longleftrightarrow$ "B is at least as hard as A"

  (i) Positive result: For solving A we can use an algorithm for B.

  (ii) Negative result: If A is hard then so is B.
- Complete problems:

  Hardest among those in the same class

  Least likely to belong to a lower class
- When finding a hard problem:
  - Analyse the borderline of hardness
  - Study special cases
  - Efficient search methods and good heuristics
  - Approximations
  - Randomised algorithms
  - Local search methods