# Algorithms & Complexity
## Lecture 8:
## Complexity Theory (Cont'd)

November 26, 2019

CentraleSupélec / ESSEC Business School

Dimo Brockhoff

Inria Saclay – Ile-de-France



# Discussion Home Exercise

## Exercise 1: What is it doing?

| | 0 | 1 | B |
|---|---|---|---|
| q0 | (q0, 0, R) | (q0, 1, R) | (q1, B, L) |
| q1 | (q2, B, R) | (q3, B, R) | STOP |
| q2 | (q4, 0, L) | (q4, 0, L) | (q4, 0, L) |
| q3 | (q4, 1, L) | (q4, 1, L) | (q4, 1, L) |
| q4 | (q4, 1, R) | (q4, 0, R) | (q1, B, L) |

1) Start with input from $\Sigma^n = \{0,1\}^n$ in q0 with head on left-most digit
2) Stays in q0 while running over the input until first B is found
3) Then move head to last digit and switch to q1
4) Now, it depends on what is read, but both cases are symmetric:

   if 0 is read, write B, go right and into q2, then write 0, go left (onto the B from before) and into q4 OR

   if 1 is read, write B, go right and into q3, then write 1, go left (onto the B from before) and into q4

# Discussion Home Exercise

**Exercise 1: What is it doing?**

| | 0 | 1 | B |
|---|---|---|---|
| q0 | (q0, 0, R) | (q0, 1, R) | (q1, B, L) |
| q1 | (q2, B, R) | (q3, B, R) | STOP |
| q2 | (q4, 0, L) | (q4, 0, L) | (q4, 0, L) |
| q3 | (q4, 1, L) | (q4, 1, L) | (q4, 1, L) |
| q4 | (q4, 1, R) | (q4, 0, R) | (q1, B, L) |

5) Hence, next step is always to go left, write back a B and go into q1

6) Now, we are back to point 4) unless we are at the (left) end of the input (=read a B) and stop

So, what is step 4 actually doing?

It shifts each cell entry one to the right, i.e., the entire input is finally moved one cell to the right!

# Discussion Home Exercise

**Exercise 2: Writing Ones Game**

aka the Busy Beaver game

Goal: Find a Turing machine with $n$ states and alphabet $\Sigma = \{1\}$ that writes as many 1s as possible before stopping

Considerations for $n = 2$:

1) Must write 1 in the beginning
2) Must also change to q1
   [otherwise: no stop]
3) In q1, need to go back to q0
   [same reason of no stop]
4) Need a stopping state

| | B | 1 |
|---|---|---|
| q0 | (q1, 1, R) | (q1, 1, L) |
| q1 | (q0, 1, L) | STOP |

# Discussion Home Exercise

**Exercise 2: Writing Ones Game**

aka the Busy Beaver game

What happens for $n > 2$?

- maximal number of 1s attainable with an $n$ states Turing Machine: a fast-growing function (not computable!)
- only optimal values for $n \leq 4$ known!

| $n = 2$ | $n = 3$ | $n = 4$ | $n = 5$ | $n = 6$ | $n = 7$ |
|---|---|---|---|---|---|
| 4 ones | 6 ones | 13 ones | 4098 ones ? | $> 3.5 \times 10^{18267}$ ones | $> 10^{10^{10^{10^{18705353}}}}$ ones |

# Course Overview

| Thu | | Topic |
|---|---|---|
| Thu, 12.09.2019 | PM | Introduction, Combinatorics, O-notation, data structures |
| Tue, 24.09.2019 | PM | Sorting algorithms I |
| Tue, 1.10.2019 | PM | Sorting algorithms II, recursive algorithms |
| Tue, 8.10.2019 | PM | Recursive and Greedy Algorithms |
| Tue, 15.10.2019 | PM | Dynamic programming |
| Thu, 31.10.2019 | AM | Randomized Algorithms and Blackbox Optimization |
| Tue, 5.11.2019 | PM | Complexity theory I |
| Tue, 26.11.2019 | PM | Complexity theory II |
| | | |
| Tue, 17.12.2019 | AM | Exam (written) |

# Motivation

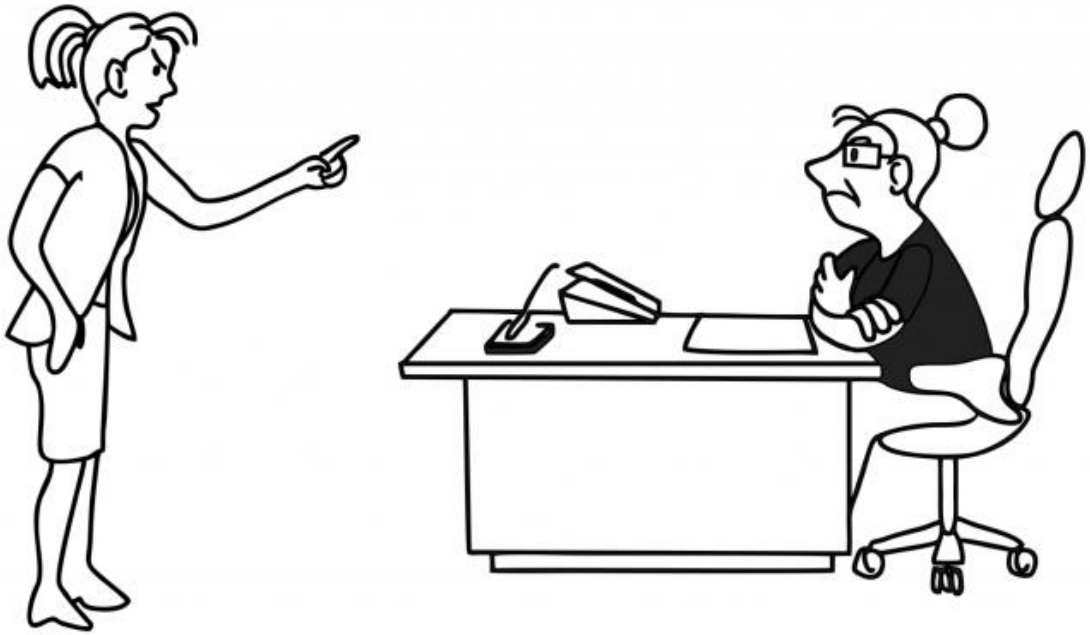
- Analyze algorithms and their runtime to solve problems
- Categorize problems according to their difficulty (for an optimal algorithm)

"I can't find an efficient algorithm, I guess I'm just too dumb."

Stefan Szeider

# Motivation:

- Analyze algorithms and their runtime to solve problems
- Categorize problems according to their difficulty (for an optimal algorithm)



"I can't find an efficient algorithm, because no such algorithm is possible!"

Stefan Szeider

# Motivation:

- Analyze algorithms and their runtime to solve problems
- Categorize problems according to their difficulty (for an optimal algorithm)

"I can't find an efficient algorithm, but neither can all these famous people."

Stefan Szeider

# Reminder: The Turing Machine (TM)

- Alan Turing (1912—1954)
- simplest ~~computer~~ computation model


**Formal definition:**

$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$

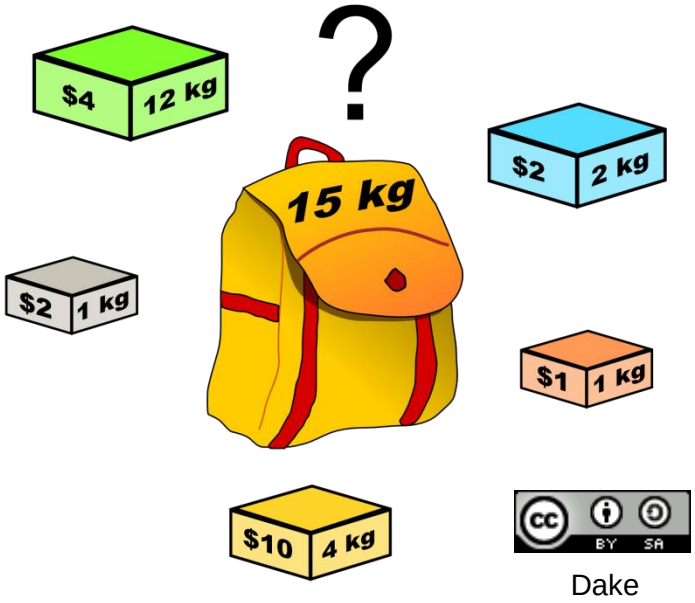

Brandon Blinkenberg

# Reminder: Different Problem Types

**Optimization problem:**

find the best solution among all feasible ones!

- KP: “find packing with maximal value”




Dake

**Search problem:**

output a solution with a given structure!

- KP: “give a packing with value V”

**Decision problem:**

is there a solution with a certain property?

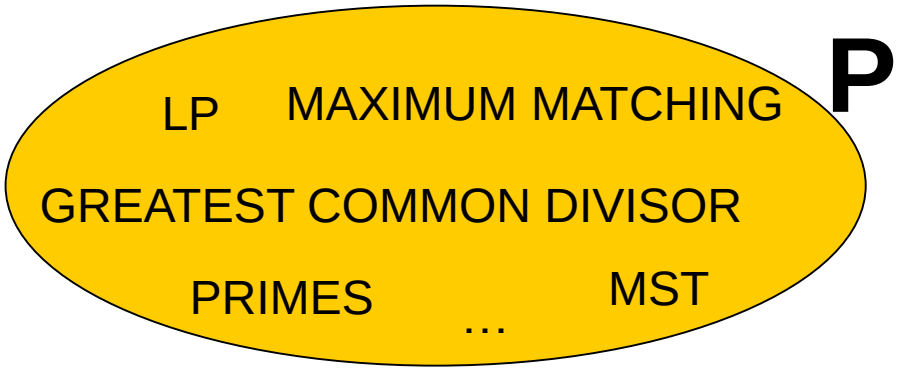
- KP: “is there a packing with value ≥V”

A decision problem is solved by a TM when it halts in an “accepting state” iff the given instance has the desired property

# The Classes DTIME(t(n)) and $\mathcal{P}$

$$DTIME\big(t(n)\big) := \{P \mid \text{P is a (decision) problem s. t. there exist an algorithm } A \text{ that solves P in time } O\big(t(n)\big)\}$$

$$\mathcal{P} = \bigcup_{k \geq 1} \mathrm{DTIME}(n^k)$$



# Nondeterministic Turing Machines

Deterministic TM (DTM) have a deterministic transition *function:*

$$\delta_{\text{det}} : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$$

Nondeterministic TM (NTM) have only a transition *relation:*

$$\delta_{\text{non-det.}} \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{R, L, N\})$$

# Nondeterminism and the Class NP

NP is the set of all problems which have polynomial time nondeterministic (!) algorithms $\mathcal{NP} = \bigcup_{k \geq 1} \mathrm{NDTIME}(n^k)$

**Intuition:**

- If I know a solution I can prove in deterministic polynomial time whether it belongs to the answer "yes" or "no"
- "Guess" the right solution and prove it in polynomial time

# Complexity Theory: Lecture Overview

- deterministic machine models
- computability
  - an example of a problem which cannot be solved by a computer: does a given Turing Machine hold after a finite number of steps?
- non-determinism and the class NP
- difficult problems:
  - the classes NP-complete, NP-hard, etc.
  - polynomial reductions
- the complexity zoo

**back to the intuition about**

**non-deterministic Turing Machines**

# Nondeterministic Turing Machines

Deterministic TM (DTM) have a deterministic transition *function:*

$$\delta_{\text{det}} : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$$

Nondeterministic TM (NTM) have only a transition *relation:*

$$\delta_{\text{non-det.}} \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{R, L, N\})$$

**Which transitions will be actually performed?**

- "lucky guesser": nondet. TM guesses the right transition
- "parallel computation": nondet. TM branches into many copies and accepts if one of the branches reaches an accepting state
- Note:
  - non-det. TMs are a theoretical construct to "analyze" problems
  - non-det. TMs can be simulated by deterministic TM

  whether this is possible in polynomial time is part of the P=NP hypothesis

# Problems in NP

## Knapsack Problem (KP)

- Guess which items to choose, check that the knapsack constraint is fulfilled, and sum up all profits

## Travelling Salesperson Problem (TSP)

- Guess a tour and sum up all edge weights

## Bin Packing (BP)

- Guess the assignment of items to bins, check that the size restrictions are fulfilled, and count the number of bins used

# the $\mathcal{P} = \mathcal{NP}$ hypothesis

**or is it a $\mathcal{P} \neq \mathcal{NP}$ hypothesis?**

# Facts about P=NP Hypothesis

- Clear: $\mathcal{P} \subseteq \mathcal{NP}$
- Not clear: $\mathcal{P} = \mathcal{NP}$
- What is the difference between, e.g., KP and PRIMES?
- For PRIMES, we know a polynomial time algorithm*, for KP, we don't
- Is KP "harder to solve" than PRIMES?
- Idea: classify the hardest problems in $\mathcal{NP}$
  - $\mathcal{NP}$-complete problems ($\mathcal{NPC} \subseteq \mathcal{NP}$)
  - Cook (1971), Levin (1973): SAT $\in \mathcal{NPC}$
  - Reductions

*Agrawal, Kayal, Saxena (2004): "Primes is in P", Annals of Mathematics, 160 (2004), 781–793
S. Cook (1971): "The Complexity of Theorem Proving Procedures", Proc. ACM symp. on Theory of computing, 151–158.
L. Levin (1973): "Universal'nye perebornye zadachi". Problemy Peredachi Informatsii 9 (3): 265–266.

# Reductions

**Idea:**

if problem A can be solved by using an algorithm for problem B, then A is not harder than B (except for a polynomial overhead)

**Polynomial Reduction** $A \leq_p B$ (Cook, 1971)

- Transform instance of A into one for B within polynomial time by a function $f$
- Use oracle for B once which computes the solution for transformed instance as solution for A
- $a \in A \iff f(a) \in B$

**Turing Reduction** $A \leq_T B$ (Karp, 1972)

- Use oracle for problem B polynomially often to compute the solution of A
- $a \in A \iff f(a) \in B$

Important: both reductions are transitive!

see http://groups.csail.mit.edu/tds/papers/Lynch/stoc74.pdf

# Example: DHC $\leq_p$ HC

**Hamiltonian Cycle**

= A cycle in a graph which visits each vertex exactly once.

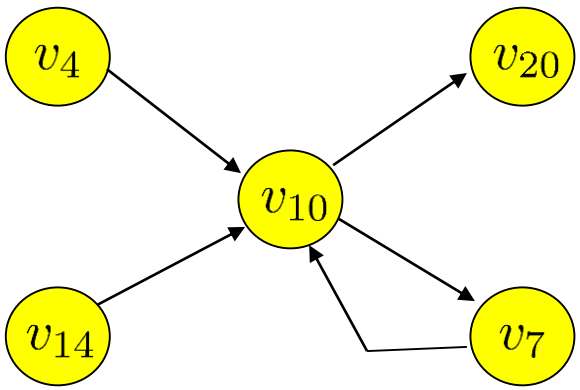
**Hamiltonian Cycle Problem (HC)**, decision version

- given an undirected graph, is there a Hamiltonian cycle?

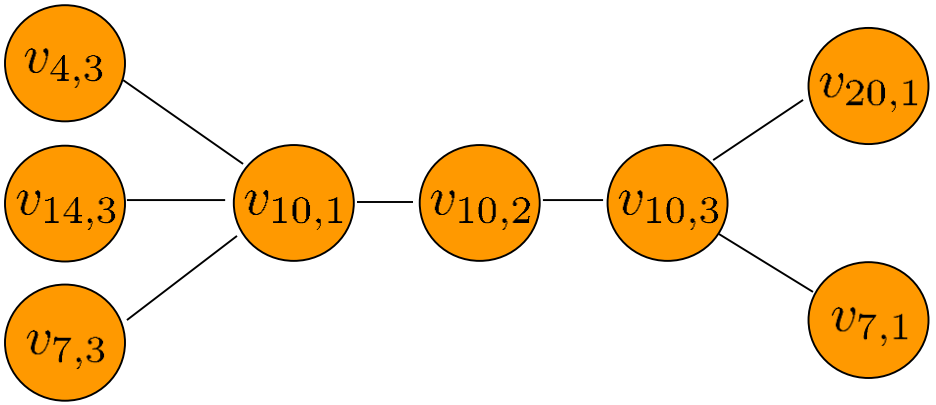
**Directed Hamiltonian Cycle Problem (DHC)**

- same for directed graphs

# Example: DHC $\leq_p$ HC




**DHC**




**HC**

- Transformation in polynomial time O(nm) possible
- Directed hamiltonian cycle in instance of DHC
  $\Longrightarrow$ Hamiltonian cycle in HC
- Hamiltonian cycle in instance of HC
  $\Longrightarrow$ order of HC is always ..., $v_{i,1}$, $v_{i,2}$, $v_{i,3}$, $v_{j,1}$, $v_{j,2}$, $v_{j,3}$, ... or
  ..., $v_{i,3}$, $v_{i,2}$, $v_{i,1}$, $v_{j,3}$, $v_{j,2}$, $v_{j,1}$, ...
  $\Longrightarrow$ take either HC or the inverted HC as solution for DHC □

Example from I. Wegener (2003): "Komplexitätstheorie", Springer

# Different Types of Polynomial Reductions

- The last example was a reduction from a special case to a general case

- Now: one slightly more complicated example

## Traveling Salesperson Problem (TSP)

- Given a set of cities and their distances
- Find the shortest path going through all cities
- Decision version: is there a (Hamiltonian) path through the graph that is shorter than a given constant?

# Example: HC $\leq_p$ TSP

**Observation:** Hamilton Cycle Problem is a subproblem of TSP

**Transformation:**

Simulate same graph for TSP as the one given for HC

- Full graph actually, but weight 1 for each edge in HC graph and weight 2 for each „non-edge“ in HC
- Asking the TSP oracle whether a weight |V| tour exists

**Correctness:**

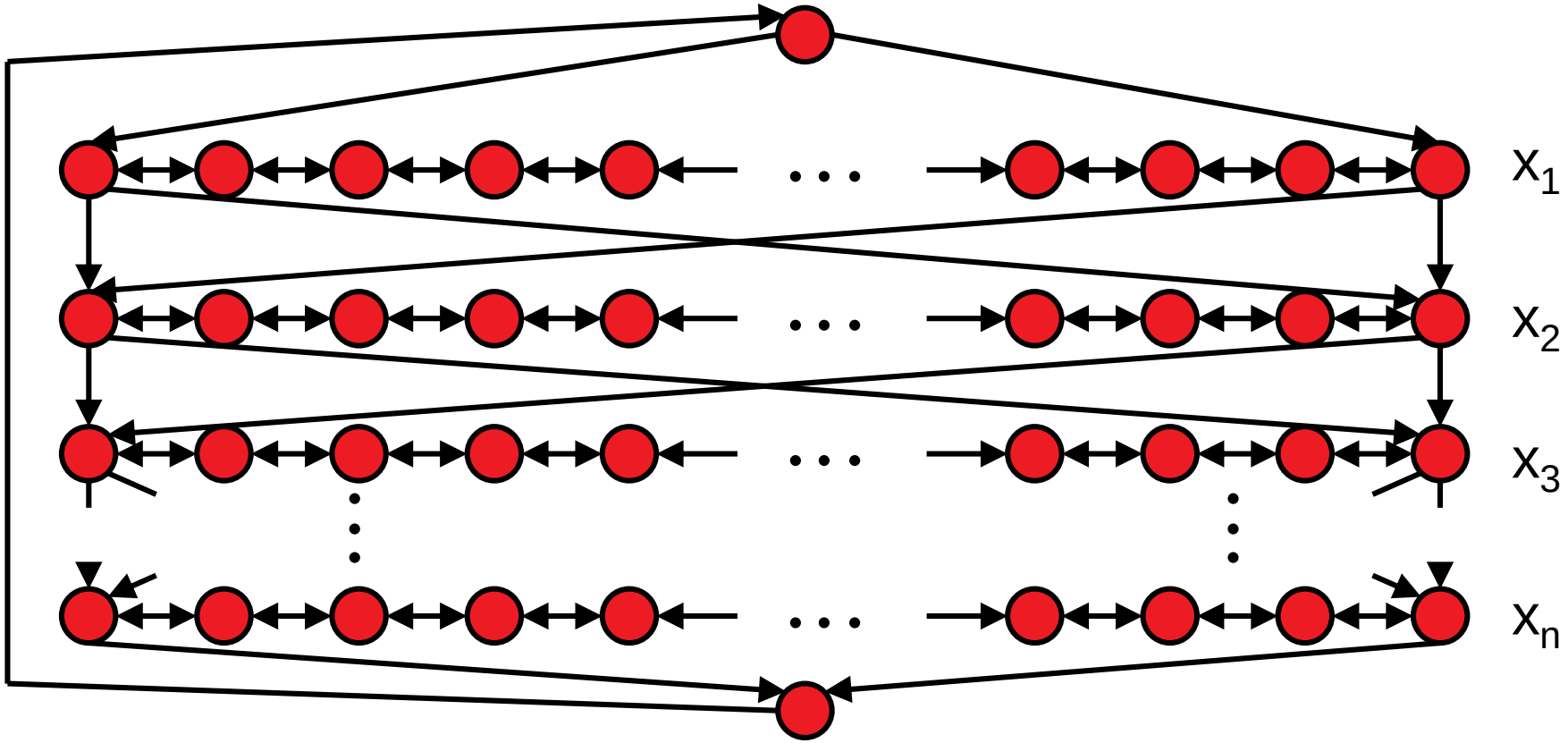
- If H is a Hamilton cycle in original graph, it is also a cycle through all cities but with weight ≤|V|
- Let T be a tour in the (transformed) TSP instance with weight ≤|V|. It cannot contain an edge with weight 2. Hence, the cycle T is also a cycle in the original HC problem.

# Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables $x_i$ and k clauses.

**Construction of DHC instance:**

- basic graph with $2^n$ many Hamilton circuits (n rows, 3k columns)
- intuition: set $x_i$ to TRUE iff its row is traversed from left to right



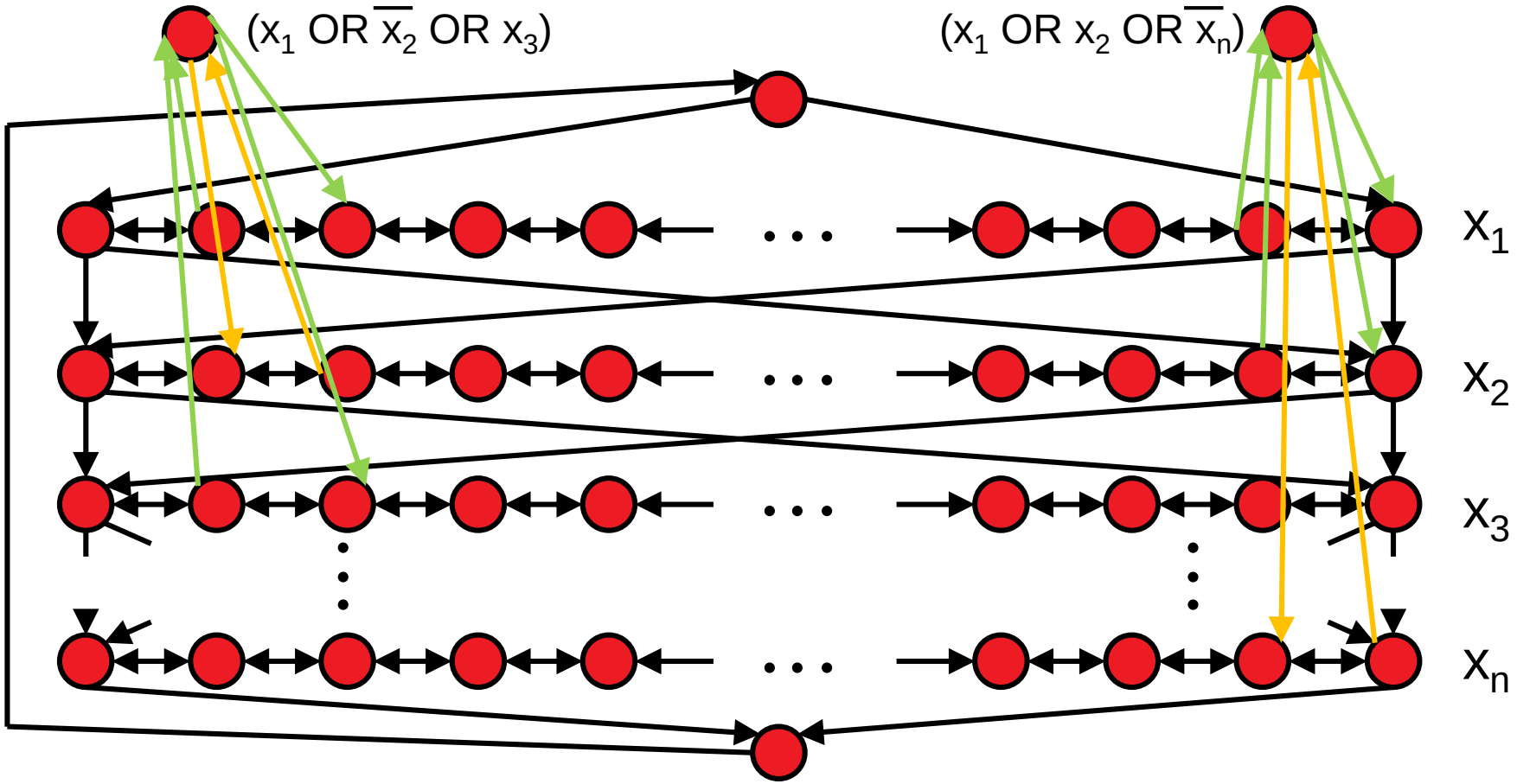

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

# Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables $x_i$ and k clauses.

**Construction of DHC instance:**

- for each clause add 1 vertex and 6 edges



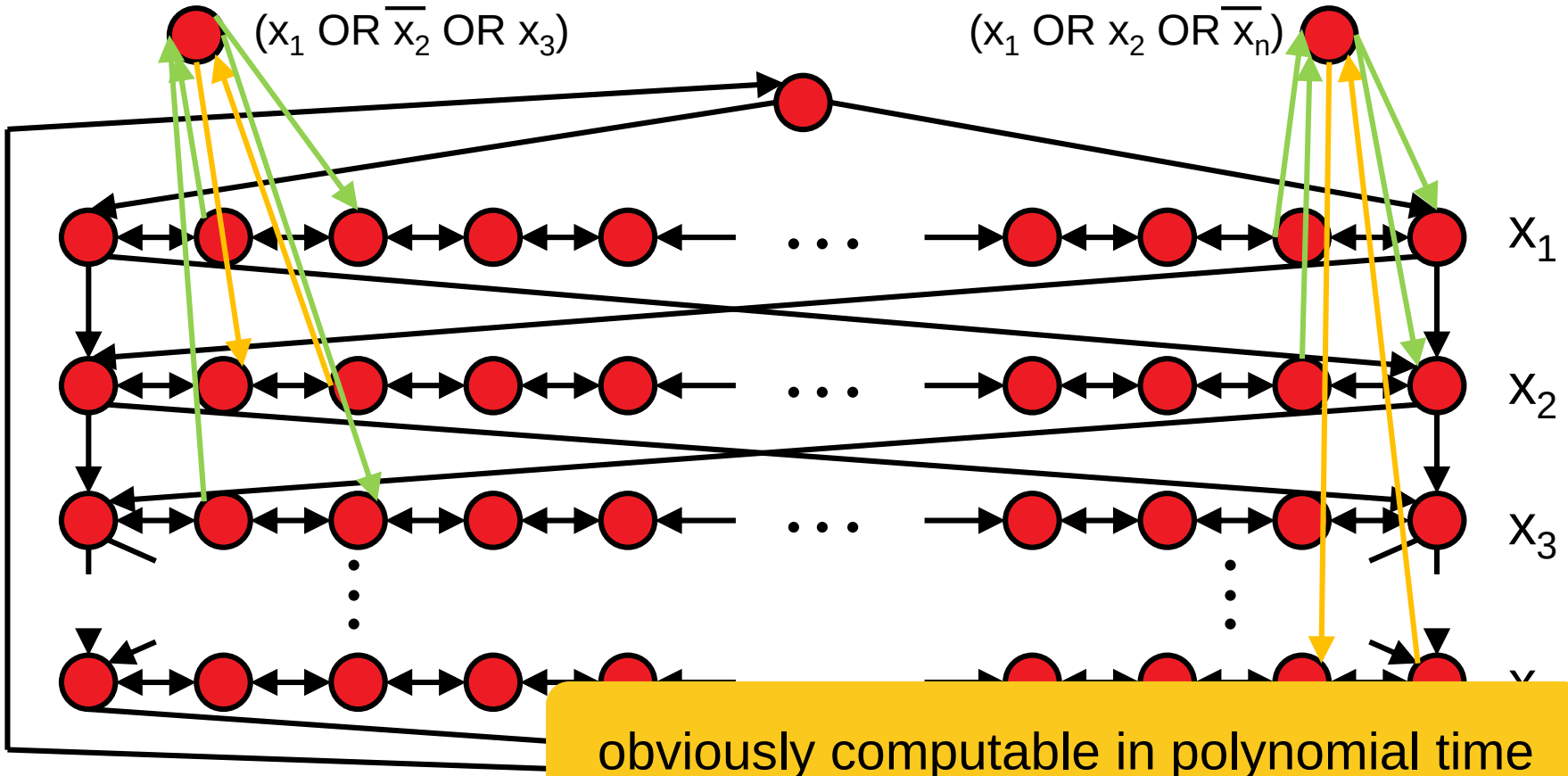

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

# Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables $x_i$ and k clauses.

**Construction of DHC instance:**

- for each clause add 1 vertex and 6 edges




obviously computable in polynomial time

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

# Proof of Correctness

**3-SAT instance is satisfiable iff corresponding graph G has Hamilton cycle!**

- let's show "⇨" first
- assume that the 3-SAT instance has satisfying assignment x*
- construct Hamiltonian cycle in G as follows:
  - if $x^*_i = 1$, traverse row i from left to right
  - if $x^*_i = 0$, traverse row i from right to left
  - for each clause $C_j$, there is at least one row i in which we are going in "correct" direction to insert the corresponding $C_j$ vertex into the tour (we do this only once per clause vertex)

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

# Proof of Correctness

**3-SAT instance is satisfiable iff corresponding graph G has Hamilton cycle!**

- now, let us see "⇦"
- assume a Hamiltonian cycle H in G
- by construction, it has to visit node $C_j$ from and to the same row
- replacing the part of H through $C_j$ by the edge in between its neighbors defines a Hamilton cycle on $G \backslash C_j$
- doing this for all $C_j$ allows to assign $x^*_i = 1$ if row i is traversed fully from left to right and $x^*_i = 0$ otherwise
- now since H traverses the clause vertex $C_j$ originally, at least one of the paths through it is traversed in "correct" order and each clause is satisfied

□

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

# The Class NPC

- $\mathcal{NPC}$: set of all $\mathcal{NP}$-complete problems
- The "hardest problems in $\mathcal{NP}$"
- A is $\mathcal{NP}$-complete if
  - A $\in \mathcal{NP}$
  - All problems $\mathrm{A}_{\mathcal{NP}} \in \mathcal{NP}$ can be polynomially reduced to A: $\forall \mathrm{A}_{\mathcal{NP}}$: $\mathrm{A}_{\mathcal{NP}} \leq_p A$

- $\mathcal{NP}$-complete problems are the hardest of the ones in $\mathcal{NP}$ in the sense that if I can solve them in polynomial time, I can solve all problems in $\mathcal{NP}$ in polynomial time

# Proving NP-completeness

How to prove that a problem A is $\mathcal{NP}$-complete?

Two possibilities:

- Either prove A $\in \mathcal{NP}$ and for all problems in $\mathcal{NP}$ that they can be reduced to A (complex, see Cook (1971)) or
- Prove A $\in \mathcal{NP}$ (simple) and a reduction from a problem B that is already known as $\mathcal{NP}$-complete to A (!)

caveat: be careful of the order in the reduction!

# The Cook-Levin Theorem

## Theorem: 3-SAT $\in$ NPC

- proven by Cook in 1971 and independently (with a slightly different proof) by Levin in 1973
- not enough time here for the detailed proof

But idea easy to understand:

- 3-SAT $\in$ NP trivial
- Given any problem p $\in$ NPC and an instance i to that problem, construct a Boolean formula which is satisfiable iff the non-deterministic TM for p accepts instance i
- Variables for states of the TM, e.g. $T_{i,j,k}$ = true if tape cell *i* contains symbol *j* at step *k* of the computation
- Polynomially many variables and Boolean statements enough because the TM runs in polynomial time

# Difference between NP-complete and NP-hard

A is NP-complete if

- $A \in \mathcal{NP}$
- $\forall B \in \mathcal{NP} : B \leq_p A$

A is NP-hard if

- $\forall B \in \mathcal{NP} : B \leq_T A$

**Implications:**

- An NP-hard problem is not necessarily a decision problem
- The search and optimization versions of an NP-complete problem are NP-hard

# Practical Implications of Reductions

The proof of NP-completeness is typically seen as a proof of difficulty:

"I did not find an efficient algorithm for my problem, maybe I am dumb?"

vs.

"I cannot find an efficient algorithm for my problem because there is none"

vs.

"I did not find an efficient algorithm for my problem but neither did all of those famous people"

# But...

Having a proof of NP-completeness or NP-hardness, does not mean that a problem is not manageable in practice:

- the average-case complexity might be reasonable
- randomized algorithms might work well
- maybe, the difficult instances are not observed

Example of success: SAT solvers

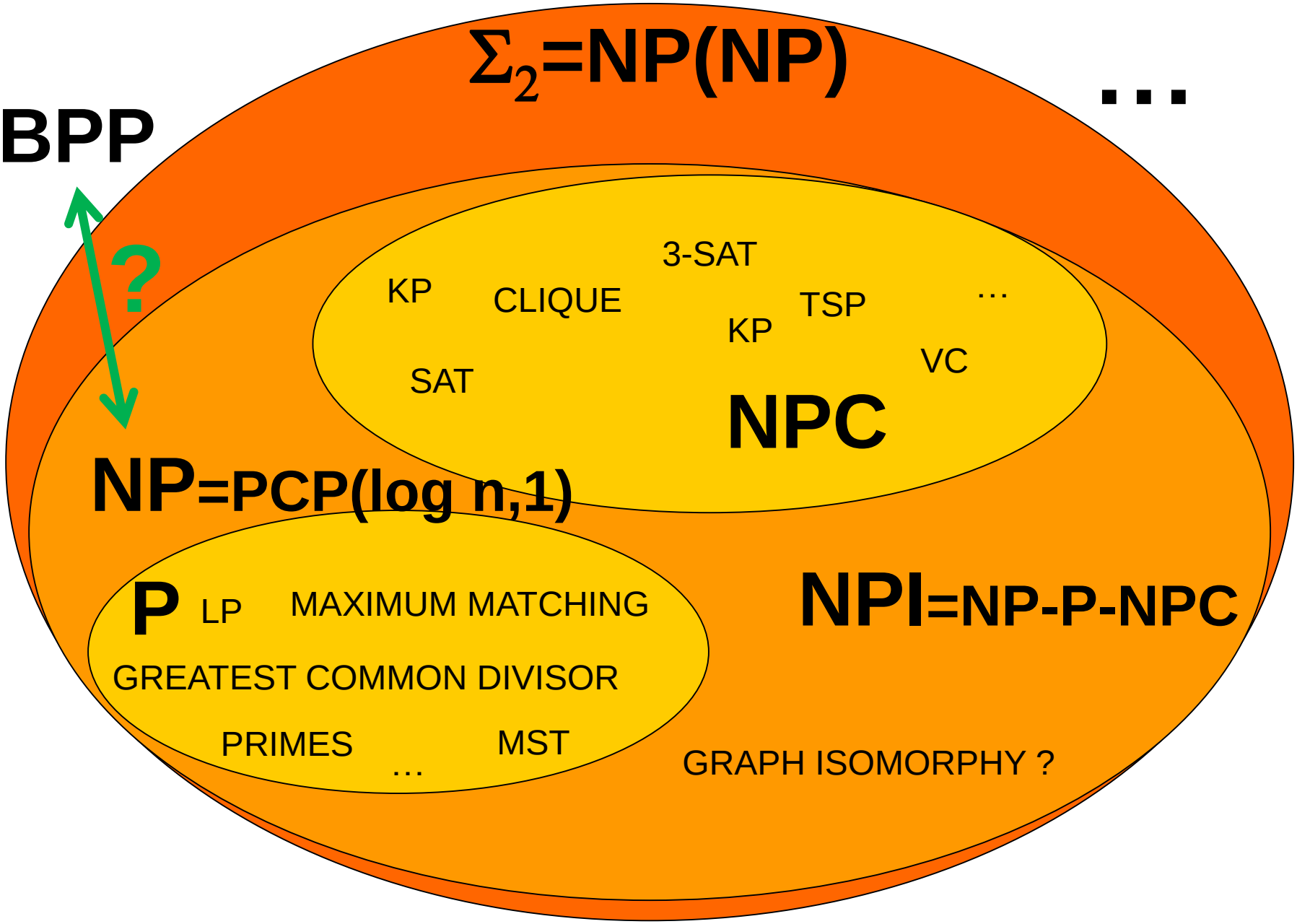
# The Famous P versus NP Problem

## Is P=NP?

- One of the 7 Millennium Prize problems selected by the Clay Mathematics Institute (worth $10^6$ $)
- first mentioned in 1956 in letter from K. Gödel to J. von Neumann
- formalized by J. Cook in his 1971 seminal paper
- solving this problem might have significant practical implications (or not)

what do you think?

# The „Complexity Zoo"



# Conclusions

I hope it became clear...

...what complexity theory is about
...what is a Random Access Machine and a Turing Machine
... how a decision and an optimization problem differ
...what are the classes P, NP, and NPC
...and that complexity theory is more involved than what we could see here ☺