Theoretische Informatik

Contents

(a) no (b) yes

Figure 1: Perfect matching?

Part I
Complexity Theory

1 Randomized Complexity Classes

1.1 Examples

1.1.1 Evaluation of Symbolic Determinants

s. Folien Hasso-Plattner-Institut

Given: a bipartite undirected graph $G = (U \cup V, E)$, $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$

Question: Does there exist a perfect matching in G? (*perfect matching:* each u_i is matched to exactly one v_i, see Figure 1)

Comment: There is a polynomial deterministic algorithm for this problem, but here we will look at a specific random approach instead.

In the adjacency matrix $A(G)$, an entry is 0 if there is no edge between u and v and a variable x_{ij} if there is.
Then, with S_n as the set of permutations over $\{1, \ldots, n\}$:

$$\det A(G) = \sum_{\pi \in S_n} \sigma(\pi) \cdot \prod_{i=1}^{n} a_i \pi(i)$$

The product is 0 if there is no matching. (σ is 1 for odd and -1 for even)

Observations:

1. The non-zero terms in $\det A(G)$ correspond to perfect matchings
2. G does contain the perfect matching $\Leftrightarrow$ $\det A(G)$ is not the zero function

$\rightarrow$ evaluate the determinant as an alternative method to answer the question.

For given values of x_{ij}, the determinant of $A(G)$ can be computed in polynomial time. For symbolic determinants, there is no polynomial time algorithm known. It is already an NP-complete problem to determine if a coefficient of a symbolic determinant is $\neq 0$. The exact value of the determinant is not necessary, only whether it is 0 or not.

Idea: choose values for x_{ij} randomly (non-neg. integers) and evaluate the resulting classical determinant. 2 cases:

1. result $\neq 0 \Rightarrow \det A(G) \equiv 0 \Rightarrow G$ contains perfect match
2. result $0 \Rightarrow$ not finished (x_{ij}s are zero of $\det A(G)$ or $\det A(G) \equiv 0$. Iterate, after a certain no. of iterations, it is unlikely that $\det A(G) \not\equiv 0$, but we will always end up in case 2

Lemma 1.1. Let $p(z_1, \ldots, z_m)$ be a polynomial in m variables, $p \not\equiv 0$. The degree in each variable is bounded by d. Let $M \geq 0, M \in \mathbb{Z}$.

$$\underbrace{|\{z_1, \ldots, z_m\} \in \{0, 1, \ldots, M-1\}^m : p(7_1, \ldots, z_m) = 0\}|}_{\text{number of zeros}} \leq m \cdot d \cdot M^{m-1}$$

Proof. by induction on m

- $m = 1$: no. of zeros of a single var. polynomial of degree $\leq d$ is $\leq d$ (fund. theorem of algebra)
- $m - 1 \rightarrow m$: a polynomial in $z_1, \ldots, z_m$ can be thought as a polynomial in z_m where the coefficients are polynomials in $z_1, \ldots, z_{m-1}$.

  If p evaluates to zero for a test point:

  - Case A: the coefficient of the highest degree term is zero

– Case B: the coefficient of the highest degree term is not zero

Case A: apply induction hypothesis to the coefficient of z_m with highest degree. Case A can arise for $\leq (m-1) \cdot d \cdot M^{m-2}$

$m-1$ tuples

Case B: polynomial of degree $\leq$ d in z_m which has $\leq d$ zeros for each $m-1$ tuple $(z_1, \ldots, z_{m-1}) \Rightarrow < d \cdot M^{m-1}$ additional zeros.

□

Back to the problem: consider the following algorithm:

1. set $m = |E|$ and M = z_m for matching cases.

   Comment: algorithm works for more general determinant but then choose m and M accordingly

2. compute set A (G) for this setting of x_{Ij} of

   if det A (G) = 0, the reply is "do not know" → iterate. After some time you might answer "G likely does not contain perfect matching".

Observation:

- algorithm never replies with YES by mistake, no false positives
- algorithm can provide false negatives.

  What is the probability for false negatives? (or of no answer, in other interpretation)

error bound for matching case d=1 it is easy to show that prob. (false negatives) ≤ 0.5 for one iteration → k repetitions →$\leq \frac{1}{2^k}$
this a algorithm belongs to the class of Monte Carlo algorithms.

1.1.2 SAT (Satisfiability)

Given: SAT formula Φ, variables $x_1, ..., x_n$, clauses $c_1, ..., c_m$
disjunction of literals $x_3 \vee x_5 \vee x_7$
$c_1 \wedge c_2 ...$

Question: does there exist a satisfying truth assignment i.e. an assignment of true and false to x_i so that all clauses are fulfilled?

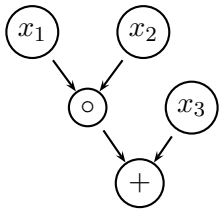

Figure 2: Algebraic circuit

Special cases of SAT:

- 2-SAT: ≤ 2 literals per clause. Solvable in polynomial time
- 3-SAT: ≤ 3 literals per clause. NP-complete

Consider the following algorithm:

1. start with an arbitrary truth assignment T for the variables $x_1, ...x_n$
2. apply the following n times.
   - If T fulfills Φ, answer YES and stop.
   - if T does not fulfill, choose a violated clause and an arbitrary literal from that clause. Flip its value (true to false or false to true)

We repeat the main step r times and then stop with answer "NO" if we haven't stopped the algorithm before.

Question: How to choose r? Does ist suffice to work with a polynomially bounded r in order to obtain a reasonable error (false negative) bound?

Answer: Already for the 3-SAT, an exponential no. of iterations is required in general. For 2-SAT there exists a polynomial bound. Choosing $r = 2^n$ leads to an error probability of $\leq \frac{1}{2}$. S. 1st exercise sheet.
$\rightarrow$ Monte Carlo algorithm for 2-SAT

1.1.3 Identity testing for polynomials

We assume that the polynomials we deal with are represented/given via an algebraic circuit

Comment: logical circuit $\wedge, \vee, /$
Analogously: algebraic circuit with the operators $+, -, \cdot$ with variables $x_1, \ldots, x_n$. It is represented as a special directed graph as follows (see Figure 2):
n source nodes $x_1, \ldots, x_n$

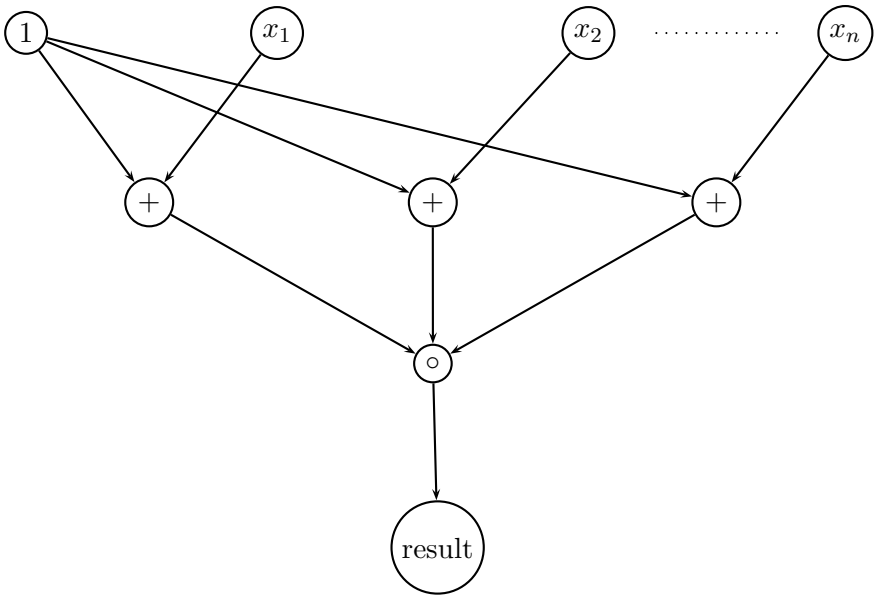

Figure 3: $\prod_{i=1}^{n}(1+x_i)$ as an algebraic circuit

Remark: Model can easily be extended, e.g. introduction of constants
Let ZERO P be the class of algebraic circuits that evaluate to a polynomial which equals the 0-polynomial.

$$C \equiv C' \Leftrightarrow C - C' \equiv 0$$

Example of an algebraic circuit:

$$\prod_{i=1}^{n}(1+x_i)$$

as an algebraic circuit:
If the polynomial is multiplied out, one has $O(2^n)$ terms.

No polynomial time algorithm is known to decide if two polynomials are identical. But randomization helps.
The following lemma along the lines of L 1.1 will play an important role

Lemma 1.2. Let $p(x_1, \ldots, x_n)$ be a polynomial of total degree at most d. Let S be a finite set of integers. Then, if $a_1 \ldots a_m$ are randomly chosen from S, then we have $Prob[p(a_1, \ldots, a_m) \neq 0 \geq 1 - \frac{d}{|S|}]$
(total degree: monomial $x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n}$ has total degree $e_1 + e_2 + \cdots + e_n$)

Observation: A circuit of size m has at most m multiplications and thus has at most degree 2^m.
$\rightarrow$ Choose the variables $x_1, \ldots, x_n \in \{1, \ldots, 10.2^m\}$. This requires $O(nm)$ random bits
$\rightarrow$ Evaluate the circuit C on $x_1, \ldots, x_n$ to obtain an output y,

- answer with "YES" if $y \neq 0$
- reject otherwise

If $C \in ZEROP$, then we will always accept.
If $C \notin ZEROP$, then we will reject with probability $\geq \frac{9}{10}$. Multiple repetitions will decrease the error probability.
The problem with the approach discussed above is that very large numbers can show up for y and the intermediary results (up to $(10.2^m)^{2^m}$ needs exponentially bits to write down)
Apply technique which is sometimes also called "fingerprinting": perform the calculations modulo K where K is an integer chosen randomly from $\{1, \ldots, 2^{2m}\}$. Instead of computing $y = C(x_1, \ldots, x_n)$, we compute $y \mod K$.
If $y = 0$, then $y \mod K$ too.
Claim: If $y \neq 0$, then with probability $\geq \delta \frac{1}{4m}$, K does not divide y (Suffices for our purposes because we can repeat our procedure $O(\frac{1}{\delta})$ times and accept only if the output is zero in all repetitions)
Assume $y \neq 0$ and let $B = \{p_1, \ldots, p_e\}$ denote the set of distinct prime factors of y.
It is sufficient to show that with probability $\geq S$, K will be a prime number not in B.
By the Prime Number Theory, we get for a sufficiently large m the number of primes $\leq 2^{2m}$ is at least $\frac{2^{2m}}{2m}$. Since y can have at most $\log y \leq Sm2^m = O(\frac{2^{2m}}{2m}$ prime factors
$\Rightarrow$ for sufficiently large m, the number of K's from $\{1, \ldots, 2^{2m}\}$, such that K is prime and is not in B is at least $\frac{2^{2m}}{4m} \Rightarrow$ a random K will have the desired properties with probability $\geq \frac{1}{4} = \delta$.

1.1.4 Prime Number Test

Given: an odd integer n

Question: Is n a prime number?
Deterministic polynomial algorithm fairly recent. Randomized algorithm: Monte Carlo.

1.2 Complexity Classes Introduced in this Chapter and their Interrelations

	probability for error/failure bounded*	probability for error/failure non-bounded*
two-sided error	BPP	PP
one-sided error	RP (Monte Carlo alg.): no false positives, false negatives possible, Co-RP (complement of RP)	NP, Co-NP
no error, but failure possible	ZPP $>$ RP $\cap$ Co-RP* (Las Vegas algorithm)	NP $\cap$ Co-NP
no error, no failure	P	

$*$ defined in this chapter

1.3 RP (Randomized Polynomial)

Key Concept: Monte Carlo Turing Machine (TM)

Definition: Let N be a polynomial, non-deterministic TM. We assume in the following that all computation on N with input x end after the same (polynomial on $|x|$) number of steps and that there are exactly 2 alternatives in each decision step.
Let L be a language, N is called a Monte Carlo TM for L if

- N fulfills the assumptions made above,
- the computations on an input of size n takes $p(n)$ time (with p polynomial),
- and if holds:
  - If $x \notin L$, then all computations of N with input x end with the answer "NO" (no false positives)
  - If $x \in L$, then at least half of the computations of N with input x end with "YES" (probability for a false negative answer is $\leq \frac{1}{2}$

Definition of the complexity class RP: The class of all languages for with there exists a Monte Carlo TM is called RP.

Comments:

- The concept of MC TM matches with the intuitive idea developed in examples.

- Replacing the bound $\frac{1}{2}$ for the probability of false negatives by some number < 1 does not lead to a new complexity class.

  $1 - \epsilon$ with $\epsilon < \frac{1}{2}$ be the probability for false negatives. K repretitions $\rightarrow (1 - \epsilon 9^K)$ ist the probability for false negatives. Running time $KO(p(n))$, polynomial for polynomial K

- RP is somewhere between P and NP
- Co-RP is the complement of RP ($\rightarrow$ complement of a language, exchange "YES" and "NO")
- not only RP is a randomized complexity class

1.4 ZPP

ZPP := RP $\cap$ Co-RP. No mistakes, no false positives, no false negatives but failure (i.e. no decision obtained) can happen. Such an algorithm is called Las Vegas algorithm.

1.5 PP

Consider the MAJSAT (Majority SAT) problem:

Given: SAT formula Φ in n variables

Question: Is it true that the majority of the 2^n truth assignments, i. e. $> 2^{n-1}$, are satisfying assignments for Φ?
It is unknown whether MAJSAT $\in$ NP (unlikely)

Definition PP: A Language L is in PP if there exists a non-deterministic polynomial TM N (fulfilling the assumptions above) such that for all inputs x we have the following: $x \in L \Rightarrow$ more than one half of the comp. of N end with answer "YES" (acceptance).
This definition is syntactical and not semantical. Not based on alg. point of view.
MAJSAT $\in$ PP. Proof easy.
MAJSAT is PP-complete

Theorem 1.3. NP $\subseteq$ PP

Proof. Let $L \in$ NP. Let N be a non-det. polyn. TM for L. We now construct a new TM N' which decides L according to the majority principle. N' is very similar to N, we add a new starting state (see Figure 4).
Input x requires $p(|x|)$ steps on N. $2^{|x|}$ different computation routes on x for N.
$2 \cdot 2^{|x|} > 2^{|x|+1}$ comp. paths of N' on $x \rightarrow$ more than $2^{|x|}$ YES paths $\Leftrightarrow x \in L \Rightarrow L \in PP$. □

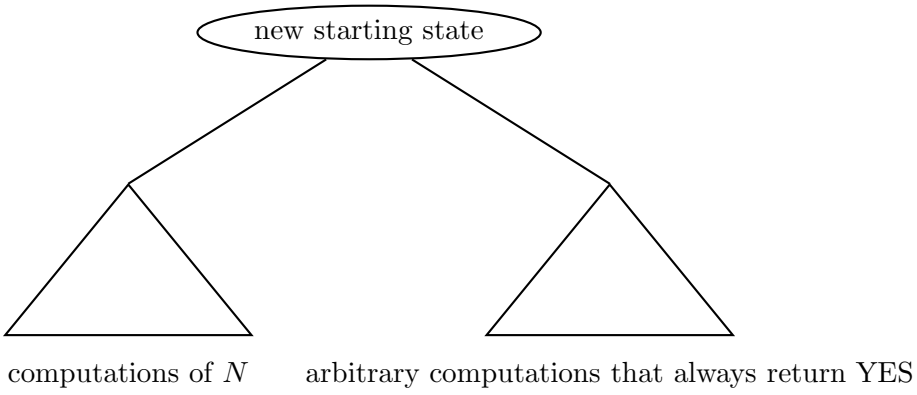

Figure 4: Majority principle TM

1.6 BPP (Bounded Error Probabilistically Polynomial)

False positives and false negatives possible ("two-sided error").

Definition: BPP contains all languages L, for which $\exists$ non-deterministic polynomial TM N (without loss of generality = wlog) + fulfilling the assumptions above, such that

- For all inputs x such that $x \in L$, at least a percentage of $\frac{3}{4}$ of all compositions of N on x end with acceptance
- For all inputs x such that $x \notin L$, at least $\frac{3}{4}$ of all compositions of N on x end with rejection.

Comments:

- $\frac{3}{4}$ not essential, can be replaced by values $> \frac{1}{2}$ and < 1
- the definition above means that the probability for false positives and false negatives is $\frac{1}{4}$
- RP $\subseteq$ BPP, Co-RP $\subseteq$ BPP (repeating MC alg. twice $\rightarrow$ error probability $\leq \frac{3}{4}$
- BPP $\subseteq$ PP (follows trivially from definition)
- It is open whether BPP $\subseteq$ NP
- It is open whether there exist BPP-complete problems
- Co-BPP = BPP
- BPP can be seen as a model for "useful" polynomial randomized algorithms

1.7 Probability and Bias

Our randomized algorithms are based on random decision, for which we need random bits/numbers, e.g. obtained through coin flip. Suppose the coin is biased (not fair), i.e. $Prob$(result of coin flip = head) $\neq \frac{1}{2}$.
This does not influence RP/BPP. Idea: show that with a biased coin, you can simulate a fair coin

Lemma 1.4. A coin with $Prob$(head) $= \rho$ can be simulated by a probabilistically polynomial (see detour below) TM in expected time $O(1)$ if the i-th bit of ρ can be determined in polynomial time in i (relevant assumption for the case ρ irrational).

Proof. Let $0.p_1p_2p_3\ldots$ be the representation of ρ in a binary system. The prob. TM (rand. alg) constructs a sequence of random bits $b_1b_2b_3\ldots$ where bit b_i is generated in the i-th step.
If $b_i < p_i$, then the TM delivers "tail" and stops. Otherwise it continues with the next bit (step $i+1$). If the TM arrives at step $i+1$, we have $b_j = p_j$ for all $j = 1, \ldots, i$. The probability for this is $(\frac{1}{2})^i \rightarrow Prob(\text{head}) = \sum\limits_k p_k \cdot (\frac{1}{2})^k = s$.
Expected running time: $\sum\limits_k k^c(\frac{1}{2})^k$ with some constant c bounded by constant $\rightarrow$ running time $O(1)$.

□

Lemma 1.5. A coin with $Prob$(head) $= \frac{1}{2}$ (i.e. a fair coin) can be simulated via access to biased coins with $Prob$(head) $= S \notin \{0, \frac{1}{2}, 1\}$ in expected time $O(\frac{1}{\rho\cdot(1-\rho)})$.

Proof. Idea: replace one flip of fair coin by sequence of tossed of biased coins. Our TM M tosses pairs of coins until it gets for the first time a pair with two different results

- coin 1: head, coin 2: tail with prob. $\rho(1-\rho) \rightarrow$ output "head"
- coin 1: tail, coin 2: head with prob. $(1-\rho)\rho$ output "tail"

Then: probability for stop in certain step $= 2\rho(1-\rho)$
$\Rightarrow$ conditioned on M is stopping in a particular step, the results head and tail for M are arising with the same probability of $\frac{1}{2}$.
Claimed expected runtime follows immediately from above □

Comment: From the above, it follows that a fair coin can be simulated in expected polynomial time by biased coin. We used polynomial worst-case time and not polynomial expected time def. of RP/BPP. However, it is easy to show that it suffices to require polynomial expected running time in the definition of RP/BPP

Detour: Concept of Pobabilistic TM

Alternative model for randomized computation
2 transition funcitons δ_0, δ_1, in each step we decide with probability $\frac{1}{2}$ each to use δ_0 or δ_1.
In the general model for a pobabilistic TM, one requires a polynomial expected running time
$\rightarrow$ RP and BPP stay the same under biased coins. Proof for the technical part from expected to worst case polyn. time not shown above, but can be done

2 The Complexity Class FP, Functional Problems and the Polynomial Hierarchy

2.1 Functional Problems

Motivation: Many problems are not of the decision problem type and have more complex answer options than "YES" and "NO".

2.2 Examples

2.2.1 FSAT

Given: SAT formula Φ

Task: If there exists a satisfying truth assignment for Φ, output one and otherwise answer "There does not exist a satisfiable formula"
Suppose for now that there exists a polynomial time algorithm for SAT (decision problem). Then there exists a polynomial time algorithm for FSAT.
Consider the following algorithm:

1. Apply the SAT algorithm A to Φ. If the answer of A is "NO", then stop and answer "Φ is not satisfiable". Otherwise with step 2.

2. (Successive fixing of variables $x_1, \ldots x_n$)
   Apply branching according to x_i, with $i = 1, \ldots, n$: Restrict the current SAT formula once to x_i = true and once to x_i = false $\rightarrow$ two new SAT formulas. Apply algorithm A to both of them. At least once we get answer "YES". Fix x_i to true of false so that the corresponding xxx SAT formula is satisfiable (leads to answer "YES" when A was accepted)

After at most n steps, we are finished and have obtained a satisfying truth assignment. $O(n)$ calls to SAT alg. $A \rightarrow$ the alg. has polynomial running time provided A has polyn. running time.

2.2.2 TSP: Travelling Salesman Problem

Given: Distance matrix $D = (d_{ij})$ with $d_{ij} \in \mathbb{Z}$

Task: Find a TSP tour with minimal length

$\text{TSP}_{\text{Decision}}$: distance matrix as above, distance bound d^*

Question: Does there exist a TSP tour with length $\leq d^*$?
Again we want to show that we could solve TSP in polyn. time, provided that TSP_{DEC} can be solved in polyn. time.
Consider the following procedure:

1. Determine the length of an optimal tour (binary search): can be bounded, e.g. by 2^q where q is the coding length of the instance. We can assume that the possible length values for the tour come from $\{0, 1, \ldots, 2^q\} \rightarrow$ apply binary search with respect to the tour length. First question: Does there exist a TSP tour with length $\leq 2^q$?

   - yes $\rightarrow$ proceed with the set $\{0, 1, \ldots, 2^{q-1}\}$.
   - no $\rightarrow$ proceed with the set $\{2^{q-1} + 1, \ldots, 2^q\}$.

   Look again at midpoint atc. After $O(\log(2^q)) = O(q)$ iterations ($\hat{=}$ one call to TSP_{DEC} algorithm, one has found the opt. value for l^*.

2. Find an optimal tour (find a tour of length l^*):

   We treat the entries d_{ij} of D separately (respectively the edges (i, j)). The idea is to check whether (i, j) is an edge needed in an optimal tour. Temporarily set d_{ij} to $l^* + 1$ (temporarily forbid the edge (i, j)). Ask the question: Does there exist a tour with length l^* with respect to the current distance matrix?

   - no $\rightarrow$exery optimal tour contains (i, j). So reset the distance value to its old value and remember that (i, j) is part of the optimal tour.
   - yes $\rightarrow$ leave d_{ij} at $l^* + 1$.

   After $O(n^2)$ steps we have found a tour of length l^*.

Overall, a polynomial number of calls to a TSP_{DEC} algorithm suffices.

2.3 FP and FNP

Let $L \in NP$. Our goal is now to define the function problem FL associated with L. For $L \in NP$, there exists a binary relation R_L which is decidable in polynomial time and such that the following holds: $\forall$ string x $\exists$ string y s. t. $(x, y \in R_L) \Leftrightarrow x \in L$. ($R_L$ decribes certificate for "yes"-instances)

Definition: Let $L \in NP$. The problem FL is the following:

Given: String x

Task: Find a string y such that $x, y \in R_L$ or reply that no such y exists.
FL is a "generalisation" of a decision problem.

Definition: FNP is the class of function problems associated with languages $L \in NP$.

Definition: FP is the class of function problems associated with languages $L \in P$.
FP is a subclass of FNP.

2.3.1 Reduction of Decision Problems → Extension to Functional Problems

	A		B
Input	x	$\xrightarrow{R}$	R(x)
			$\downarrow$
Output	S(z)	$\xleftarrow{S}$	z

Definition: Let A and B be function problem. A is said to be reducible, if the following holds: $\exists$ string functions R and S (both computable in polynomial time, alternatively one could require that both ate computable within logarithmic space), such that: If x is the input string of an instance I_A of problem A, then $R(x)$ is the input string of an instance I_B of problem B. If z is a correct output for the instance I_B, then $S(z)$ is a correct output for the instance I_A.

2.3.2 Completeness for Function Problem Classes

Definition: A function problem A is called complete for the class FC of function problems if

- $A \in FC$ and
- All problems in FC are reducible to A.

Special case FNP-completeness.
One can easily show that FSAT is FNP-complete. Similarly, one can show many results for FNP-completeness for function problems which are associated with NP-complete decision problems.

2.4 Special Cases and Examples

Special class of function problems: those for which there always exists a y so that $(x, y) \in R_L$. (The corresponding decision problem then is trivial.)

2.4.1 Factorisation of Natural Numbers

Given: $N \in \mathbb{N}$

Task: Decompose N into its prime factors.
The corresponding decision problem is trivial. It is open whether there exists a polynomial time algorithm for factorisation.
The example above motivates to define the class of total functions.

Definition: We call a problem L in FNP total if $\forall$ strings x $\exists$ string y such that$(x, y) \in R_L$.
TFNP (sub)class of all function problems associated with total functions.

2.4.2 Example 2

Given: n numbers $a, \ldots, a_n \in \mathbb{Z}^+$ such that $\sum_{i=1}^{h} a_i < 2^n - 1 \quad (\star)$

Task: Find $I, J \subseteq \{1, \ldots, n\}, I \neq J$ such that $\sum_{i \in I} a_i = \sum_{j \in J} a_j$ (alternatively $I \cap J = \varnothing$) or answer that no such I and J exist (One can show that due to $(\star)$, the no case does not occur[1] $\rightarrow$ example for a total function)
Without $(\star)$, already the corresponding decision problem is NP-complete (subset sum, partition,...).

2.4.3 Happy Net

Given: undirected graph $G = (V, E)$. Edge weights $w_e \in \mathbb{Z}$ for all $e \in E$. A state function S is a function $V \rightarrow \{+1, -1\}$. A vertex $i \in V$ is called happy with respect to a state function S if the following holds:

$$S(i) \cdot \sum_{i,j \in E} S(j) w_{ij} \geq 0$$

Happy Net Problem: Find a state (function) S sucht that all vertices are happy.

Claim: There always exists an S such that all vertices are happy $\rightarrow$ total function
No polynomial time algorithm for HAPPYNET is known.

Proof. $\Phi(s) := \sum_{(i,j) \in E} s(i) s(j) w_{ij}$
Suppose vertex i is not happy with respect to S i.e. $S(i) \cdot \sum_{(i,j)} S(j) w_{ij} < 0 = -\delta$ where $\delta > 0$.

[1] pigeon hole principle of Schubfachschluss – there are 2^9 non-empty subsets and only $2^n - 2$ possible values for the sums

Under the assumption that $w_e \in \mathbb{Z}$, one can deduce $\delta \geq 1$ (for rational case adapt).
We move from S to a new state S' such that $S'(j) := S(j)$ and $S'(i) := -S(i)$, $\forall j \neq i$ (flip the state of i)
One could easily check that $\Phi(S') = \Phi(S) + 2\delta \Rightarrow$ the potential function increases in each such step by ≥ 2.
This suggests the following algorithm:

1. start with an arbitrary initial state
2. as long as there are unhappy vertices, choose one and flip its state

Since ϕ can take only values from $[-W, W]$ with $W = \sum_{(i,j)\in E} |w_{ij}|$ and since ϕ increases by ≥ 2 after each flip, the above algorithm terminates with a solution to HAPPYNET in a finite number of steps.
The running time $O(W)$ is not polynomial in input size. □

2.4.4 Example 4: Confer Problem 6 in exercises

2.5 DP: Difference Polynomial

Definition: $L \in DP \overset{\text{Def.}}{\Rightarrow} \exists$ languages $L_1 \in$ NP and $L_2 \in$ Co-NP such that $L = L_1 \cap L_2$.

Remark: DP $\neq$ NP $\cap$ Co-NP (DP is much larger than NP $\cap$ Co-NP)

2.6 Motivating Example: Exact-TSP $\in$ DP

4 variants:

1. TSP_{DEC}.
   Given: $n \times n$ matrix D, bound c^*.
   Question: Does opt. tour have length $\leq c^*$?
2. EXACT-TSP
   Given: $n \times n$ matrix D, value c^*.
   Question: Does opt. tour have length c^*?
3. TSP-COST
   Given: $n \times n$ matrix D
   Task: Determine length of optimal tour
4. TSP
   Given: $n \times n$ matrix D
   Task: Find an optimal tour

All four versions are polynomially equivalent.

2.6.1 Example 1: EXACT-TSP

EXACT-TSP $\in DP$ because the question whether $\exists$ tour with length $= c^*$ can be split up into a $\geq c^*$ question of "Co-NP type" and $\leq c^*$ question of "NP type"

Theorem 2.1. EXACT-TSP $\in$ DP-complete.

Proof. (sketch)

1. EXACT-TSP $\in$ DP already argued
2. EXACT-TSP $\in$ DP-complete:

   Reduction from SAT-UNSAT, starting point is the classical reduction from 3-SAT to the Hamiltonian cycle (path) problem.

   We construct two graphs G and G' from the two 3-SAT formulas Φ and Φ' such that G respectively G' contain a Hamiltonian path $\Leftrightarrow$ Φ respectively Φ' are satisfiable.

□

2.6.2 Example 2: SAT-UNSAT

Theorem 2.2. SAT-UNSAT $\in$ DP-complete

Given: SAT formulas Φ and Φ'

Question: Is Φ satisfiable and is Φ' non-satisfiable?

1. SAT-UNSAT $\in$ DP

   $L_1 = \{(\Phi, \Phi') : \Phi \text{ is satisfiable } \} \in$ NP, since SAT $\in$ NP (use Cook's Theorem)

   $L_2 = \{(\Phi, \Phi') : \Phi' \text{ is not satisfiable } \} \in$ Co-NP, since Co-SAT $\in$ Co-NP

   SAT-UNSAT $= L_1 \cap L_2$

2. Let $L \in$ DP. We need to show that L can be reduced to SAT-UNSAT.

   $L \in$ DP, i.e. $\exists L_1 \in$ NP, $L_2 \in$ Co-NP such that $L = L_1 \cap L_2$

   $\Rightarrow \exists$ reduction R_1 from L_1 to SAT

   $\Rightarrow \exists$ reduction R_2 from L_2 to SAT

   We build up a reduction R with $R(x) = (R_1(x), R_2(x))$ from L to SAT-UNSAT such that $R(x)$ is yes-instance of SAT-UNSAT $\Leftrightarrow$ $R_1(x)$ is satisfiable and $R_2(x)$ is not satisfiable $\Leftrightarrow$ $x \in L_1$ and $x \in L_2 \Leftrightarrow x \in L$

The basic idea of the reduction from 3-SAT to Hamiltonian path:

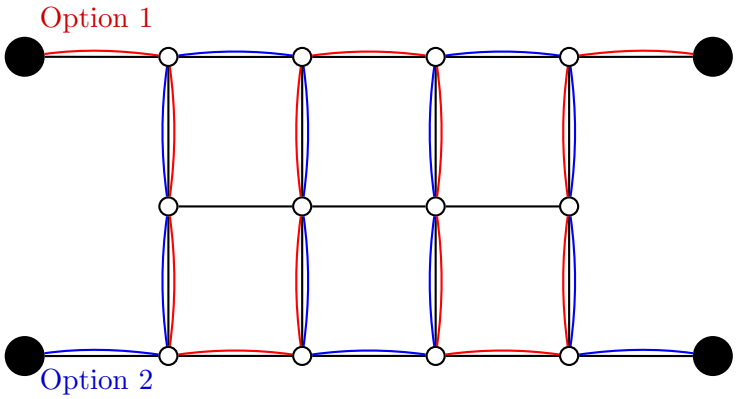

Figure 5: Consistency Gadget

Given: 3-SAT formula Φ with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$.
Construct a graph $G = R(\Phi)$ such that G contains a Hamiltonian path $\Leftrightarrow$ Φ is satisfiable.
Our construction with different types of gadgets. One choice gadget per variable.
Consistency gadget
the inner nodes only appear in the gadget $\rightarrow$ 2 ways of inserting this gadget into a Hamiltonian path. Only one can be used – implements exclusive or. Short-hand notation:
How about the clauses?
Clause gadget
We assume that each clause contains exactly 3 literals. The consistency gadgets are used to make sure that we cannot cheat and use x_i = true in clause c_j and x_i = false in clause c_k.
Our graph G contains n copies of the choice gadget, one for each variable, connected in series
Example $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$
All $3m$ triangle nodes and the last node of the series connection of choice gadgets and the new vertex 3 are linked by all possible edges (complete subgraph) + introduce new vertex 2 and link it to 3
This concludes the construction $\Phi \rightarrow G = R(\Phi)$.
Major claim: Φ is satisfiable $\Leftrightarrow$ G contains a Hamiltonian path. Proof: exercise (This will be a path between vertices 1 and 2).
The novel part of our construction will be such that G and G' contain a so-called broken Hamiltonian path[2] and that a Hamiltonian path exists iff the corresponding formula is satisfiable.
The broken Hamiltonian path corresponds to an almost satisfying truth assignment, i.e. an assignment that satisfies all clauses except one. To achieve this, we modify our SAT formula: add a new variable z and add z to all clauses (in unnegated form) and add the new clause $\bar{z}$ (this can be transformed to 3SAT-instance).
Given an instance (Φ, Φ') of SAT-UNSAT, we apply the machinery from above to construct two graphs G and G' which contain broken Hamiltonian paths and have a Ham.

[2] $\hat{=}$ two disjoint paths that cover all nodes, "one edge missing"

path $\Leftrightarrow$ corresponding formula is satisfiable.
Identify 1 of G and 2 of G', 2 of G and 1 of G'
Based on this graph we now define distances:
Set $d_{ij} = \begin{pmatrix} 1 & \text{if } \{\text{i, j }\} \text{ is an edge in } G \text{ or } G' \\ 2 & \text{if } \{\text{i, j }\} \text{ is not an edge in } G \text{ but } i \text{ and } j \text{ are nodes in } G' \\ 3 & \text{otherwise} \end{pmatrix}$

Question: What is the length of an optimal tour? Two cases:

1. Φ and Φ' are satisfiable $\rightarrow$ G and G' contain Ham. paths P_1 and P_2 which together form tour for our graph, all tour edges have weight 1. Cost of $\tilde{n}$ is the number of nodes.

2. Φ and Φ' are not satisfiable. 2 broken Ham. paths. Cost to repair the 2 breaks is $+1$ (edge of cost 2 instead of cost 1). $+2 \rightarrow$ total $\tilde{n} + 3$ (edge of cost 3 instead of 1)

3. Φ is satisfiable and Φ' is not satisfiable. optimal value $\tilde{n} + 2$

4. Φ is not satisfiable and Φ' is satisfiable. optimal value $\tilde{n} + 1$

Use $d^* = \tilde{n} + 2$ for the EXACT-TSP instance.
Many further NP-hard opt. problems lead to DP-complete problems when one looks at the corresponding EXACT version. This shows that DP is a kind of natural class for EXACT problems.
Another such class are *critical problems*.

2.7 Critical problems

Critical SAT, Critical Hamiltonian path, Critical 3-Colorability are DP-complete

2.7.1 CRITICAL SAT

Given: SAT formula Φ

Question: Is it true that Φ is not satisfiable, but by removing an arbitrary clause, we end up with a satisfiable formula?

2.7.2 Critical Hamiltonian path

Given: Undirected graph $G = (V, E)$

Question: Is it true that G does not contain a Hamiltonian path, but by adding an arbitrary new edge, we end up with a Hamiltonian path.

2.7.3 UNIQUE SAT

Given: SAT formula Φ

Question: Does there exist a unique satisfying assignment? (only one possible)
It is unknown whether UNIQUE-SAT $\in$ DP-complete.

2.8 A step towards the polynomial hierarchy

Another way to look at DP: We are allowed to call once a SAT oracle and to call once a Co-SAT oracle ($\equiv$ 2 SAT oracle calls).
$\rightarrow$ Idea: allow a *polynomial* number of calls to a SAT oracle
$\rightarrow$ leads to the concept of a so-called Oracle Turing Machine

Definition: A TM $M^?$ with an oracle is a TM with a special string (the so-called query string) which corresponds to a call to the oracle and 3 special states (in addition to the standard TM concept):

- $q_?$ state which corresponds to call of oracle
- q_{YES} state corresponding to "YES" answer from the oracle call
- q_{NO} state corresponding to "NO" answer from the oracle call

Let $A \subseteq \Sigma^*$ be a language. $M^?$ with oracle A runs like a classical TM except for state transitions from the query state.
$M^A(x)$: apply the oracle TM $M^?$ with oracle A to input x. Instead of using single languages we also allow classes of languages.
One of the classes we will be interested in will be what we denote by $\mathrm{P}^{\mathrm{SAT}}$ (generalization of DP). We consider oracle TM which use a polynomial number of steps (= ordinary steps + calls to the SAT oracle).
As (3SAT) SAT is a representative for NP-complete problems (hardest problems in NP), $\mathrm{P}^{\mathrm{SAT}}$ is the same as P^{NP}
Analogously, once can define $\mathrm{FP}^{\mathrm{NP}}$.

Question: Are there $\mathrm{FP}^{\mathrm{NP}}$-complete problems?

2.8.1 MAX OUTPUT

Given: non deterministic TM N, with input 1^n (string of n times 1). N behaves such that it stops for this input after $O(n)$ steps and produces a binary string of length n as output.

Task: Determine the largest output of N (seen as a binary number)

Theorem 2.3. MAX OUTPUT $\in$ FP$^{\text{NP}}$-complete.

Proof. (Sketch)

i. MAX OUTPUT $\in$ FP$^{\text{NP}}$.

Use binary search, iteratively ask whether $\exists$ output $\geq x$, (x in decimal system from $0, \ldots, 2^n - 1) \rightarrow$ in $O(n)$ steps we get the value of the largest output (correpsonds to calls to an NP oracle $\rightarrow$ MAX OUTPUT $\in$ FP$^{\text{NP}}$

ii. MAX OUTPUT $\in$ FP$^{\text{NP}}$-COMPLETE.

Let F be a function mapping string to strings with $F \in$ FP$^{\text{NP}}$. $\Rightarrow \exists$ oracle TM $N^?$ which can evaluate F in a polynomial number of steps, i.e. $\text{M}^{\text{NP}}(x) = \text{M}^{\text{SAT}} = F(x)$.

To show the above completeness, we need a reduction from F to MAX OUTPUT.

input $F \xrightarrow{R} R(x)$

solution $f(R(x)) \xleftarrow{S} F(x)$

(a) R and S are computble within logarithmic space

(b) $\forall$ string x, $R(x)$ is the input for a MAX OUTPUT instance

(c) S applied to the output for the MAX OUTPUT instance with input $R(x)$ delivers $F(x)$

Construction of R

We need to construct a non-deterministic TM N and a string 1^n. We set $n = p^2(|x|)$ where $|x|$ is the encoding length of x and $p()$ is the polynomial bound for the number of steps performed by M$^{\text{SAT}}$. Informally, n is set large enough to allow us to simulate M$^{\text{SAT}}$.

N starts with the input 1^n and generates generates the beginning x as a string and then simulates M$^{\text{SAT}}$ on x. This simulation will work in a deterministic way. Except for oracle calls, N can do the same as M$^{\text{SAT}}$. Suppose we arrive at the the first oracle call (question whether a SAT formula Φ_1 (Φ_i) is satisfiable, in general at the i-th oracle call). N guesses the answer to the question and sets $z_1 = 1$ ($z_i = 1$) if the guesses answer is yes and $z_1 = 0$ ($z_i = 0$) otherwise. If $z_i = 0$, N proceeds with the state q_{NO}.

If $z_i = 1$, N guesses a truth assignment T_i for the variables in Φ_i. Next, N checks whether T_i satisfies Φ_i. If this is the case, N proceeds with state q_{YES}. Otherwise, N "gives up", i. e. outputs 0^n and stops (non-successful computation).

In case that N does not stop prematurely, it writes the string $z_1, z_2, z_3, \ldots$ on the output band and fills the remaining positions with zeros in order to get an output

string of length n and finally adds the output of $\mathrm{M}^{\mathrm{SAT}}(x)$, i. e. $F(x)$ (refer to this as successful computation).

N is a non-deterministic TM. Some of the successful computations are wrong computations of $\mathrm{M}^{\mathrm{SAT}}(x)$ (Φ_i can be satisfiable and N can proceed with $z_i = 0$. For $z_i = 1$, the simulation is always correct if successful).

Key observation: The successful computation that delivers the largest output corresponds to a correct simulation of $\mathrm{M}^{\mathrm{SAT}}(x)$.

Proof of the observation: Suppose that in a successful computation which delivers the max. output we have $z_j = 0$, but Φ_i satisfiable (one source of mistake) and let j be minimal with this property. There needs to exist another successful computation which coincides with the considered up to the j-th oracle call, but then shows $p_j = 1$ and finds a satisfying assignment T_j and such that all further oracle calls are simulated correctly.

Contradiction to the fact that the original successful computation provided the max. output.

Concluding, N solves the MAX OUTPUT problem. N can be constructed within logarithmic space, S is trivial since $F(x)$ has been written on the output band.

□

2.8.2 MAX-WEIGHT SAT

Given: A SAT formula with variables $x_i, \ldots x_n$ and clauses $C_i, \ldots, c_m$. Each of the clauses c_j is assigned an integer weight w_j.

Task: Find a truth assignment such that the total weight (sum of weights) of all satisfied clauses is maximized (satisfiability not required).

Theorem 2.4. MAX-WEIGHT SAT $\in \mathrm{FP}^{\mathrm{NP}}(x)$-complete.

Proof. (Sketch)

i. MAX-WEIGHT SAT $\in \mathrm{FP}^{\mathrm{NP}}(x)$: use binary search and SAT oracle $\rightarrow$ provides the largest possible weight and then do variable fixing as in FSAT

ii. We use the techniques from the proof of Cooks theorem (This proof constructs from a non-deterministic TM and an input of 1^n a SAT formula $\Phi(N, n)$ such that each satisfying truth assignment for Φ corresponds to (successful) computations of N

□

2.9 Polynomial...

Missing: 27 and 29 October

Theorem 2.5.

Theorem 2.6.

Theorem 2.7.

Theorem 2.8.

Theorem 2.9.

Theorem 2.10.

Theorem 2.11.

Theorem 2.12.

Comment: another example for $\Sigma_i P$-complete problems comes from multilevel programming (bilevel, three-level)

- each of the decision makers have his own decision functions, constraints and variables
- The leader starts to make a decision, in order to obtain a function value for his objective function, he depends on his follower's objective function
- assume that the follower chooses the worst situation for the leader
- you can show that these problems are complete, depending on the number of decision makers you introduce

Theorem 2.13. If there existed a PH-complete problem, the polynomial hierarchy would collapse at a finite level.

Proof. Assume L PH-complete $\Rightarrow \exists i \geq 0 : L \in \Sigma_i P \Rightarrow$ each language $L' \in \Sigma_{i+1}$P can be reduced to L. Since all levels of PH are closed with respect to reductions, we get $L' \in \Sigma_i \mathrm{P} \Rightarrow \Sigma_{i+1}\mathrm{P} = \Sigma_i$ P. □

It is unknown whether PH-complete problems exist.

Theorem 2.14. PH $\subseteq$ PSPACE (without proof here)

Conjecture: PH $\neq$ PSPACE

Remarks:

1. Quantified Boolean formula problem $Q_1x_1Q_2x_2\dots Q_kx_k \quad \Phi(x_1,\dots x_n)$: log. formula free of quantors. $Q_i \in \{\forall, \exists\}$ is a known PSPACE-complete problem
2. When defining Σ_i P and Π_i P via alternating oracle TMs, one needs to assume that i, the number of quantors is not part of input $\rightarrow$ in PSPACE, there is no restriction on the number of quantors
3. Since there exist PSPACE-complete problems, the existence of PH-complete problemts would imply the collapse of the polynomial hierarchy.

Theorem 2.15. BPP $\subseteq \Sigma_2$P

Proof. Let $L \in$ BPP $\Rightarrow \exists$ TM M with calculations of length $p(n)$ (p polynomial) for inputs of length w sucht that M decides L by majority vote (prob. for false answers $\leq \frac{1}{4}$ For each input x of length n denote by $A(x) \in \{0,1\}^{p(n)}$ the set of accepting computations. We can assume that $\forall x \in L|A(x)| \geq 2^{p(n)}(1 - \frac{1}{2^n}$ (repeat such that error probability $\leq 12^n \forall x \notin L|A(x)| \leq \frac{1}{2^n} \cdot 2^{p(n)}$

□

Definition $a, b \in U$, $aXORb$ (bitwise XOR operation)

Observations

- $aXORb = c \Leftrightarrow cXORb = a$
- $(aXORb)XORb = a \rightarrow$ "XOR b" provides a 1 to 1 mapping
- $a \in U$ fixed, $r \in U$ random $\rightarrow aXORr$ random

Definition: translation $t \in U$, $A(x)XORt := \{aXORt | a \in A(x)$ translation of $A(x)$ by $t\}$
Theorem BPP $\subseteq \Sigma$ P
$A(x) \oplus t := \{a(+)t | \hat{\subset} A(x)\}$ (bitwise xor)
From properties of $\oplus$ we get $|A(x) \oplus t| = |A(x)| \forall t \in U$
be a random sequence of $p(n)$ translations ($p^2(n)$ random bits)
Now consider $b \in U$, arbitrary but fixed. We say that b is covered by $t_1, \dots, t_{p(n)}$ if $\exists j \in \{1, \dots, p(n)\}$ such that $b \in A(x) \oplus t_j$.

Question: What is the probability that b is covered by $t_1 \dots t_p(n)$? $b \in A(x) \oplus t_j \overset{\Leftrightarrow}{prop(i)of} \oplus b(+ = t_j \in A(x)$ (t_j random, right sight: prop (iii) – random has same distribution as t_j)
Recall that we can assume $\forall x \in L|A(x)| \geq 2^{p(n)}(1 - \frac{1}{2^n}, \forall x \notin L|A(x)| \leq \frac{1}{2^n}$:...

Since $x \in$, we have that prob$(b \notin A(x) \oplus t_j) \leq \frac{1}{2^n} = 2^{-n}$ (follows from ([])). $\rightarrow$ probability that b is not covered by any the t_j is $\leq 2^{-np(n)}$
$\Rightarrow$ each element of U is not covered with prob $\leq 2^{-np(n)}$
$\Rightarrow$ the prob. that then $\exists$ element in U that is not covered is $\leq 2^{-n(pn)}.|U| = 2^{-(n-1)p(n)} << 1$
$\Rightarrow$ A sequence of $p(n)$ random translations covers all of U with high probability.
Assume now $x \notin L$. Now $A(x)$ contains an exponentially small part of U. This implies that for large enough n there cannot exist a sequence of p(n) translations that cover all of U.
Resume: $x \in L \Leftrightarrow \exists$ sequence of $p(n)$ translations that cover U
$L = \{x : \exists T = (t_1, \dots t_{p(n)}) \in \{0,1\}^{p(n)}$
$\forall b \in U : \exists j \in \{1, \dots, p(n)\}$ sucht tha $b \oplus t_j \in A(x)\}$
The last $\exists$ quantor can be reformulated to $(b(x)t_1 \in A(x)) \vee (b \oplus t_2 \in A(x) \vee \cdots \vee$
can be checked in polyn. time $\rightarrow \Sigma_2$ char. results $\rightarrow L \in \Sigma_2 P$

Remark: BPP is closed with respect to complement $BPP \subseteq \Sigma_2 P \cap \Pi_2 P$

3 Complexity of Counting (Problems)

We are interested in the number of solutions to a problem, or more precisely in the complexity of these counting problems.

3.1 Examples

3.1.1 #SAT

Given: SAT formula Φ

Task: Find the number of satisfying truth assignments for Φ
There exist numerous similar problems which come from problems where the associated decision problem is NP-complete such as #HAMILTONIAN CYCLES/PATHS etc.

3.1.2 #MATCHING (PERMANENT)

Given: Bipartite graph $G = (U \cup V, E), U = \{u_1, \dots, u_n\}, V = \{v_1, \dots, v_n\}$

Task: Find the number of perfect matchings (1 to 1 assignments in U and V) in G.
Perfect matching corresponds to $\Pi \in S_n$, where S_n is a permutation on $\{1, \dots, n\}$
Associate with G its adjacency matrix $A(G) = a_{ij}$ with $a_{ij} = \begin{cases} 1 & u_{i,v_j} \in E \\ 0 & \text{otherwise} \end{cases}$
$\Pi \in S_n$ describes a perfect matching in $G \Leftrightarrow \{u_i, v_{\Pi(i)}\} \in E \forall i \in \{1 \dots, n\} \Leftrightarrow a_{i\Pi(i)} = 1 \forall i \in \{1 \dots, n\} \Leftrightarrow \prod_{i=1}^{n} a_{i\Pi(i)} = 1$
Remark: perm(A) looks similar to det(A)

htb

Figure 6: Reduction

$\Rightarrow \sharp$ perfect matchings $= \sum_{\Pi \in S_n} \prod_{i-1}^{n} a_i \Pi(i)$ (permanent of A, perm(A), permanent is defined for general matrices) $= \sum_{\pi \in S_n} (-1)^{sgn(\Pi)} \prod_{i=1}^{n} a_{i\Pi(i)}$
There exists an efficient polynomial time algorithm for computing $\det(A)$ but most probably not for $\text{perm}(A)$ even in the 0 -1 case
Note that deciding whether there exists a perfect matching is an easy problem.

3.1.3 Example 3

Given: graph with m edges and n vertices

Task: Find the number of subgraphs of G (overall there are 2^m subgraphs) which contain a path from 1 to n.
Of interest in reliability questions.

3.2 #P

Let Q be a polynomial balanced binary relation which is checkable in polynomial time. Now we associate the following counting problem to Q:

Given: x

Task: Find the number of y with $(x, y) \in Q$.
#P is the class of counting problems which are associated with such relations Q.
All counting problems considered in our chapter are from the set $\sharp$P

Question: Do there exist $\sharp$P-complete problems?. Attention: we need an appropriate reduction notion here.
Counting problems are special functional problems.
For defining $\sharp$P-completeness there exist (among others) the following two approaches:

1. Use the reduction notion defined for functional problems. We need a function S which determines the number of solutions of problem A when given the number of solutions of problem B.

   A very popular subclass of such reductions (parsimonious) arises when the number of solutions is preserved (S = ident)

2. In formal idea A is a function problem if it is ♯P-complete, if problem is in ♯P and if the existence of a polynomial time algorithm for the computation of f would imply ♯P = FP.

   For formal version, extend the notion of an oracle TM M to functional problems. We provide M with access to an oracle for $f : \{0,1\}^* \to \{0,1\}^k$ (for $f : \{0,1\}^* \to \mathbb{N}$ using binary format. M has access to the language $L = \{(x,i) : f(x)_i = 1\})$ $(i-th$ bit) For f refer to FP f as the set of functions which can be computed with a polynomial oracle TM with access to f.

   Function f is ♯P-complete if $F \in \sharp$ P and if $\forall y \in \sharp$P $y \in \mathrm{FP}^f$ (Obviously $f \in$ FP, $\mathrm{FP}^f = \mathrm{FP}$)

Theorem 3.1. Let $f \in$ #P-complete and $f \in \mathrm{FP} \Rightarrow \mathrm{FP} = \#\mathrm{P}$.

Theorem 3.2. #SAT $\in$ #P-complete.

Proof. (Sketch) Consider the proof of theorem of Cook: provides a reduction from an arbitrary language L onto SAT $\to$ it provides a function f computable in polynomial time. $f : \{0,1\}^* \to \{0,1\}^*$
$f(x) \in \mathrm{SAT} \Leftrightarrow x \in L$ ("$\in$ SAT": yes instance of SAT)
The proof does not only provide f but also a method which transforms a certificate for $x \in L$ to a certificate for $f(x) \in$ SAT (set of satisfiable SAT formulas)
$\to$ We have a bijection between yes instances $\to$ parsimonious reduction.
obviously #SAT $\in$ #P

□

Theorem 3.3. #HAMILTONIAN PATHS/CYCLES $\in$ #P-complete.

Theorem 3.4. PERMANENT (#MATCHING) $\in$ #P-complete.

Proof. (Sketch, some steps omitted, not required in the exam) Our goal is a #P-completeness result for comp. permanent of 0-1 matrices. We take a detour via more general matrices.

1. For 0-1 matrixes A: perm(A) = # perf. matchings in graph $G(A)$, bipartite graph associated with A.

2. For matrices A with entries from $\{0, \pm 1\}$: one can easily prove that perm(A) = $|\Pi \in S_n : \prod_{i=1}^{n} a_{i\Pi(i)} = 1\}| - |\Pi \in S_n : \prod_{i=1}^{n} a_{i\Pi(i)} = -1\}|$ (cardinalities)

   ($\to$ 2 #SAT calls suffice)

3. Let A be an $n \times n$ matrix with integers as entries. Consider A as a weighted adjacency matrix of the directed, complete graph K_n (but directed and continues loops) with $V = \{v_1, \ldots, v_n\}$

   Use the cycle representation of $\Pi \in S_n$.

Figure 7: Weights of cycle covers

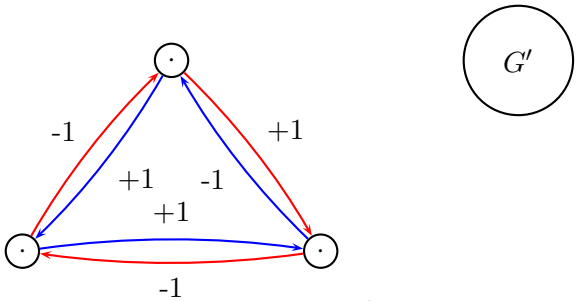

Figure 8: Weights

	1	2	3	4
Π	3	1	2	4

Observation: each permutation $\Pi \in S_n$ corresponds to a cycle cover in $G(A)$ (a set of directed cycle in $G(A)$ such that each vertex is contained in exactly one cycle)

Next define the weight of the cycle cover as the product of the weights of the included arcs.

$\rightarrow$ perm(A) = sum of the weights of all cycle covers. With this construction one can show that computing the permanent for an integer matrix x is in $\mathrm{FP}^{\#\mathrm{SAT}}$

4. Reduction from #3SAT to PERMANENT

   In the remaining part of the proof, edges with weight 0 are not drawn (all involved graphs are complete). The weight of a cycle cover which contains such an edge is 0 (see definition of weight above). We will first allow parallel edges (we can get rid of this). See Figure 7

   Consider the following graph G which splits into the following 2 parts (2nd: vertex disjoint): Figure 8

   Observe that there exist exactly 2 cycle covers in the left subgraph which have non-zero weight. One of them has weight 1 and the other one the weight -1.

   To any cycle cover of weight w in G', we get two cycle covers of the whole of G, one with weight w and the others with weight $-w \rightarrow$ the permanent of $A(G)$ is 0.

   In order to prove Valiant's theorem, we will reduce the #P-complete #SAT problem to PERMANENT.

   Let Φ be a 3SAT formula with n variables $x_1, \ldots, x_n$ and m clauses. First, we will show how to construct an integer matrix (or equivalently a weighted directed graph G) such that $\mathrm{perm}(A(\tilde{G})) = 4^{3m} \cdot \#\Phi$, with $\#\Phi$ denoting the number of satisfying truth assignments for Φ. The ocmputation of the integer matrix is #P-complete.

   This matrix will not be a 0-1 matrix, we will take care of this issue later.

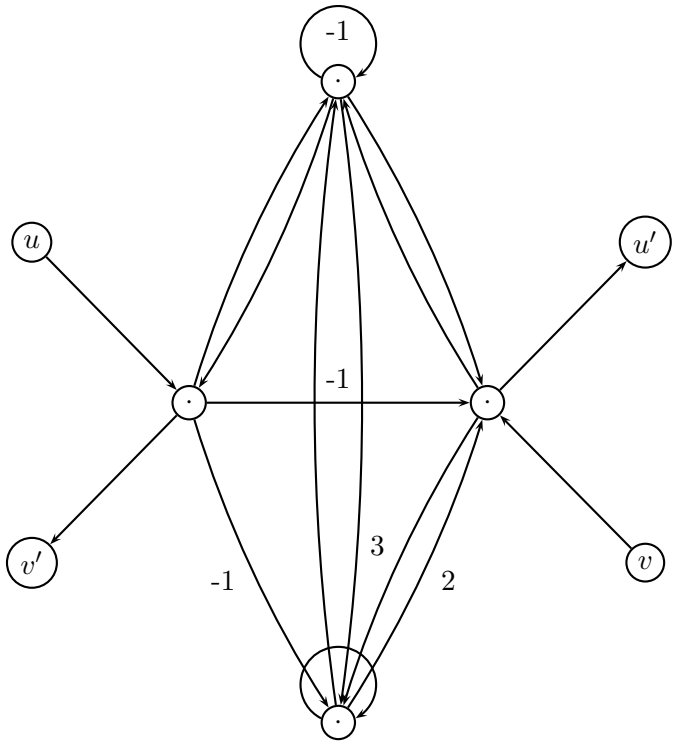

Figure 9: XOR gadget

Similarly to the the idea in the simple example above, we will make use of negative weights to ensure that the contribution of cycle covers that do not correspond to satisfying assignments cancel out. For each satisfying assignment we want to have the contribution $4^{3m} \to$ leads to $(\star)$.

To construct $\tilde{G}$, we combine three types of gadgets (variable gadgets, clause gadgets, XOR gadgets). There is a variable gadget per variable, a clause gadget per clause

XOR Gadget: see Figure 9. Edges with no indicated weight have weight 1.

$$D = \begin{pmatrix} 0 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 3 & 0 \end{pmatrix}$$

One can explicitly check that

- a $\operatorname{perm}(D) = 0$
- b1 $\operatorname{perm}(D') = 0$ where D' results from D by deleting the first column and first row.
- b2 last instead of first

b3 if first and last rows and columns are deleted

c perm $(D'') = 4$ where P'' results from deleting the first row and last column (or the last row and first column)

Note: the first and last columns are the ones which contain zero entries.

The idea behind the XOR gadget is that, given a weighted directed graph H, we want to ensure that for some pair of directed edges (u, u') and (v, v'), exactly one of these edges is contained in any cycle cover that contributes to the final sum.

Every cycle cover of H of weight w that uses exactly one of the edges (u, u') and (v, v') is mapped to a cycle cover in a graph H' whose total weight is $4w$ (i.e. the set of covers that enter the gadget at u and exit at u' or enter at v and exit at v') while all cycle covers of H' have total weight 0 (details as exercise).

Variable gadget: has both internal (only contained in this gadget) and external edges (will be connected via XOR gadgets to other edges in the graph). We split the external edges into "true" and "false" edges

The variable gadget as only 2 cycle tokens (correspond to selling this variable to 0 or 1)

0: using all "false" external edges, 1: using all "true" external edges + all other vertices will be covered by loops.

Variable gadgets Each external edge of a variable gadget is associated with a clause in which this variable appears. It is associated with a true edge, if the variable appears in this clause unnegated, false otherwise.

Clause gadget Each external edge is connected via an XOR gadget with the external edge of the variable gadget which corresponds to this external edge of the clause gadget.

Observation: the only possible cycle covers of the clause gadget are those which omit at least one external edge.

Moreover, it can be checked that for each proper subset of the external edges there exists a unique cycle cover of weight 1 that contains them.

Overall:

Analysis of the construction: Since each variable gadget has exactly two cycle covers (one for $x_i = 1$, one for $x_i = 0$), there is a one-to-one correspondence between assignments to the variable $\overrightarrow{x} = x_1, \ldots, x_n$ and the cycle covers in $\tilde{G}$ which covers the variable gadgets. For every assignment $\overrightarrow{x}$ let $C_{\overrightarrow{x}}$ denote the set of cycle covers in $\tilde{G}$ that cover the variable gadgets according to $\overrightarrow{x}$.

That means: we cover variable x_i's gadgets using the "true" external edges if $x_i = 1$ and cover it by the "false" external edges otherwise.

Let $w(\overrightarrow{x})$ denote the total weight of assignments in $C_{\overrightarrow{x}}$. It suffices to show that $w(x) = 4^{3m}$ if x is a satisfying assignment and $w(\overrightarrow{x} =)0$ else.

To prove this, we argue as follows: By the properties of the XOR-gadget and the remaining construction we know the following:

If $x_i = 1$, then in cycle covers in $C_{\overrightarrow{x}}$, the corresponding external edge has to be omitted in the gadget of every clause that contains x_i positively and it has to be included for clauses which contain x_i negatively.

For $x_i = 0$, reverse include/omit. (All other covers do not contribute to final sum).

Since every cover of the clause gadget has to omit at least one external edge, we see that unless every clause has a literal that evaluates to true in $\overrightarrow{x}$ (i. e. unless $\overrightarrow{x}$ satisfies Φ), the total weight of covers in $C_{\overrightarrow{x}}$ will be zero. If $\overrightarrow{x}$ does satisfy Φ, then the total weight will be 4^{3m} (since $\overrightarrow{x}$ determines a unique cycle cover for all clause gadgets that passes through XOR gadget exactly $3m$ times).

5. It remains to reduce to the 0-1 case. The essential question is how to deal with the -1 entries. Also to be answered: how to deal with entries with 1, $1 \neq 1$ entries The transformation works in 2 steps. First, we reduce from the integer case to $\{0, \pm 1\}$ and then to $\{0, 1\}$.

   Suppose we have an edge with a weight being a power of 2, say 2^k. Then we can replace this edge by a path of k edges (consisting of new nodes, not connected of something else), each of weight 2. An edge of weight $2^k + 2^{k'}$ by two parallel paths with weights 2^k and $2^{k'}$. By using binary expansion $\rightarrow$ set of paths with corresponding weights. The total number of nodes in these paths is quadratic in the number of bits needed to represent the original weight. $\rightarrow O(L^2)$ new nodes.

   To get rid of the negative weights, we use modular arithmetic.

   For $0, \pm 1$ matrixes $n \times n$ A, we know that $\text{perm}(A) \leq n!$ and $-n! \leq \text{perm}(A)$

   We choose $M = n^2$ and then take everything mod $2^M + 1$

   $-1 \equiv 2^M (mod 2^M + 1)$, so we can replace the -1 weights by 2^M weights. Apply construction before (construction of polynomial size) $\rightarrow$ subgraph of size $0(M^2)$

□

3.3 Approximately Counting the Number of Solutions

Remark: One can also deal with the question of approximately counting the number of solutions. α-approximation for $0 < \alpha < 1$. Algorithm A delivers an α-approximation for $f : \{0,1\}^t \rightarrow No.$ if $\alpha \cdot f(x) \leq A(X) \leq \frac{f(x)}{\alpha} \forall x$, where $A(x)$ is the result of A. There are #P-complete problems for which even determining an α-approximation for the number of solutions is a hard problem for constant $\alpha > 0$. There are problems for which α-approximation counting is easier/easy.
For the 0-1 PERMANENT problem, there exists FPRAS (fully polynomial randomized approximation scheme).

$\rightarrow$ algorithm which for given $\varepsilon > 0$ and $\delta > 0$we get an $(1-\varepsilon)$ approximation for the function f we want to determine which works correctly with probability $1-\delta$ (it might fail with probability δ) and has polynomial running time $p(n, log\frac{1}{\delta}, \log\frac{1}{\varepsilon})$, with n being the coding length of the input.

Comment: If P = NP, then each problem in # P would have a FPRAS and even an FPTAS (deterministic, no mistake, see last part of course).

Question: How powerful is counting?
Since problems in #P can be solved within polynomial space (e.g. by lexicographic enumeration), #P is not more mighty than PSPACE.
Now compare PH and #P.

Theorem 3.5. (Theorem of Toda, without proof)
$PH \subseteq P^{\#P} = P^{\#SAT}$.

$\rightarrow$ We can solve each problem from the polynomial hierarchy by making use of a polynomial number of calls to a #P(-complete)/#SAT oracle.
Informally: "Counting is more powerful than the polynomial hierarchy"

Remark: one can show that if #P = FP $\Rightarrow$ NP = P and PH = P.

Comment: There is a relation between PP[3] and #P. $\rightarrow$ interest in the leading (first) bit of the numbers of accepting computations (when including leading zeros).
Idea: What about if we are interested only in the last bit? $\rightarrow$ We only focus on the parity: even/odd $\rightarrow$ motivation for +P (spoken: parity P, odd P).

3.4 +P

A language L is in +P, if there exists a polyn. TM M s.t. $\forall$ strings x we get $x \in L \Leftrightarrow$ # of accepting computations of M on x is odd.
Equivalently: $\exists$ polynomial balanced relation R s.t. $x \in L \Leftrightarrow$ # of y's s.t. $(x, y) \in R$ is odd.

3.4.1 Example: +SAT

Given: SAT formula Φ

Question: Is the # of satisfying truth assignments odd?

Theorem 3.6. +SAt is $\oplus$P-complete

(Proof relies on standard techniques.)
Similarly we get that +HAM PATH/CYCLES is +P-complete

[3]Underlying question of PP[4]: are more than half of the computations accepting?

Question: Is +MATCHING +P-complete?
No: this problem is solvable in polynomial time by computing the determinant of the matrix associated with our bipartite graph.

Theorem 3.7. +P is closed with respect to complement

Theorem 3.8. NP $\subseteq$ RP$^{\oplus P}$

Proof. We will construct a polynomial Monte Carlo algorithm for SAT which uses a $\oplus$SAT oracle. Let a SAT formula Φ in variables $x_1, \ldots, x_n$ be given.

Definition: Let $S \subseteq \{1, \ldots, n\}$
Hyperplane η_S: The boolean expression which requires that an even number of variables with indices from the set S have value true.
Let $y_0, y_1, \ldots, y_n$ be new variables. η_S can be obtained as follows: Take the conjunction of the following clauses $(y_0), (y_n)$ and for all $i \in \{1, \ldots, n\}$, we add the clause $\tilde{C}_i$ with
$\tilde{C}_i : y_i \Leftrightarrow (y_{i-1} \oplus x_i)$ if $i \in S$
$\tilde{C}_i : y_i \Leftrightarrow y_{i-1}$ if $i \notin S$
$\tilde{C}_i$ can be transformed to the usual SAT normal form.
Our Monte Carlo algorithm now works as follows:
Start with $\Phi_0 = \Phi$ (the given SAT formula)
For $i := 1$ to n do:
generate randomly a set $S_i \subseteq \{1, \ldots, n\}$ and set $\Phi_i := \Phi_{i-1} \wedge \eta_{S_i}$. Apply the +SAT oracle to Φ_i.
If $\Phi_i \in$ +SAT, then Φ is satisfiable. Answer "yes" and stop.
Otherwise, answer "probably no", "no".

Claim 1: the algorithm above is indeed a MC-algorithm
Proof of Claim 1:

1. We don't have false positives. If Φ_i has ≥ 1 satisfying assignment, Φ has at least one.

2. Probability for false negative?

   Claim 2: Prob(false negatives) $\leq \frac{7}{8}$ (repeat 6 times $\rightarrow \leq \frac{1}{2}$

   Proof of Claim 2:

   Central observation: If the number of satisfying assignments for Φ is between 2^k and $1^{k+1} (0 \leq k < n)$ with $k \in \mathbb{N}$, then the probability that Φ_{k+1} has exactly one satisfying assignment is $\geq \frac{1}{8}$

   Proof of the central observation:

   Let T be the set of satisfying assignments for Φ. Suppose $2^k \leq |T| \leq 2^{k+1}$. We say that two truth assignments coincide on a hyperplane η_S if they both satisfy η_S or they both violate it.

Let $t \in T$ (fixed). Consider $\hat{t} \in T$. Prob(t and $\hat{t}$ coincide on the first $k+2$ hyperplanes $(\eta_{S_1}, \ldots, \eta_{s_{k+2}})) = \frac{1}{2^{k+2}}$ (since this would mean that all $k+2$ sets $S_1, \ldots, S_{k+2}$ contain an even number of variables, where t and $\hat{t}$ disagree and all these events are independent and arise with probability $\frac{1}{2}$.

Summing up over all $\hat{t} \in T \backslash \{t\}$. $\rightarrow$ probability that t agrees with some $\hat{t} \in T \backslash \{t\}$ on the first $k+2$ hyperplanes $\leq \frac{|T|-1}{2^{k+1}} < \frac{1}{2}$ by assign. on $|T|$ on the first $k+2$ hyperplanes is $\geq \frac{1}{2}$.

The probability that t satisfies all first $k+2$ hyperplanes is $\frac{1}{2^{k+2}}$.

We know that if it satisfies them, then with probability $\geq \frac{1}{2}$, it is the only one that does (the fact that t satisfies the first $k+2$ hyperplanes does not affect the probability that it disagrees with the rest of T).

$\Rightarrow$ with probability $\geq \frac{1}{2^{k+3}} = \frac{1}{2} \cdot \frac{1}{2^{k+2}}$, t is the unique satisfying assignment for Φ_{k+2}. This holds for all $t \in T$ and T has $\geq 2^k$ elements $\Rightarrow$ The probability that such a suitable element exists is at least $2^k \cdot \frac{1}{2^{k+3}} = \frac{1}{8}$.

If the number of satisfying assignments for Φ is $\neq 0$, then there exists a $k < n, k \in \mathbb{N}_0$, such that this number is between 2^k and $2^{k+1} \Rightarrow$ At least one of the $\Phi_i' s$ will have probability at least $\frac{1}{8}$ to be satisfied by a unique truth assignment and thus an odd number.

□

4 Interactive Protocols

4.1 Introduction and Definitions

two roles/persons who interact:

- the prover
- the verifier

They communicate/exchange messages or information. The prover wants to convince the verifier that his proof is correct. In the end, it is the task of the verifier to either accept or reject the proof.

Key Decision: How much power do the prover and the verifier have?

4.1.1 Variant 1

Assume both prover and verifier are deterministic (i. e. act deterministically)
Exchange of messages:
function $f, g : \{0,1\}^* \rightarrow \{0,1\}^*$ for $k \in \mathbb{N}_0$ (which may depend on the input length) and x as the input known both to prover and verifier. k round interaction between prover and verifier is a sequence of strings $a_1, \ldots, a_k \in \{0,1\}^*$ such that

- $a_1 = f(x)$
- $a_2 = g(x, a_1)$
- $a_{2i+1} = f(x_1, a_1, \ldots, a_{2i})$ for $2_i < k$
- $a_{2i+2} = g(x_1, a_1, \ldots, a_{2i+1})$ for $2_{i+1} < k$

The output at the end of the interaction is denoted by $out_f(f, g)(x)$ defined as $f(x, a_1, \ldots, a_k)$ and we assume that the output is from $\{0, 1\}$ (0: rejection, 1: acceptance).

Deterministic proof system: We say that a language L has a k round deterministic interactive proof system if there is a deterministic TM V (verifier) that on input $x, a_1, \ldots a_k$ runs in polynomial time in the $|x|$ and can have a k round interaction with any function P such that

- (Completeness) $x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^* out_V(V, P)(x) = 1$
- (Soundness) $x \notin L \Rightarrow \forall P : \{0,1\}^* \to \{0,1\}^* out_V(V, P)(x) = 0$

$\to$ dIP (deterministic interactive protocol) is the class of all languages with a k round deterministic proof system where $k(n)$ is polynomial in n.

Comments: No restrictions are made on the power of prover P (intuitively, this makes sense as a wrong proof needs to be rejected also if the proof is very clever) $\to$ we could eliminate the dependence of P on x (exercise).
One can prove that dIP does not give as anything new.

Lemma 4.1. dIP = NP

(proof: exercise)
This negative experience suggests to allow the verifier to act like a randomized algorithm $\to$ the verifier is allowed to make mistakes, but the error probability should be limited in a meaningful way.

4.1.2 Variant 2

This leads to the complexity class IP.

- Verifier: has randomised polynomial time available (rand. polyn. algorithm).
- Prover: has exponential computing time available, deterministic has exponential computation time available, deterministic suffices. It already suffices to provide the prover with PSPACE power.

Again, we can come up with the concept of a k round exchange protocol but now *random bits* enter the game for the verifier.
In the IP concept, the random bits of the verifier are assumed to be known only to the verifier ("private random bits" as opposed to so-called public random bits). Again, the prover and the verifier exchange messages, but now the mesages of the verifier are allowed to depend also on the verifier's random bits. The output of the protocols is now a random variable.

Probabilistic verification and the class IP For an integer $k \geq 1$ (that may depend on $|x|$), we say that a language L is in IP$[k]$ or IP(k) if there exists a probabilistic polynomial TM (or alternatively a BPP algorithm) V such that we can have a k round interaction with a function $P : \{0,1\}^* \to \{0,1\}^*$ such that

- (Completeness) $x \in L \Rightarrow \exists P : \{0,1\}^* \to \{0,1\}^* : \mathbb{P}[out_V(V,P)(x) = 1] \geq \frac{3}{4}$
- (Soundness) $x \notin L \Rightarrow P : \{0,1\}^* \to \{0,1\}^* : \mathbb{P}[out_V(V,P)(x) = 1] \leq \frac{1}{4}$[5]

Comments:

- The probability is with respect to r, the vector of random bits of V, but this is typically omitted from the notation.
- IP $= \bigcup$ IP $[n^c]$ all what we get our new concept with a polynomial number of rounds
- The bounds $\frac{3}{4}$ and $\frac{1}{4}$ can be made arbitrarily close to 1 and 0 respectively by the same "boosting" technique as for BPP

One can show the following:

Lemma 4.2. The class IP defined above remains unchanged if we replace $\frac{3}{4}$ by $1 - 2^{-n^s}$ and $\frac{1}{4}$ by 2^{-n^s} for any fixed constant $s > 0$.

(Proof uses repetitions, Chernoff bounds)

Comments

- One can easily show that providing the prover with randomisation does not provide anything new. We still get IP (by averaging: deterministic prover that makes the prover accept with the same probability)
- Since the prover can use an arbitrary function, it could use, in principle, even unbounded computation time or even compute undecidable functions. However, one can show that given any verifier V, we can compute the optimum prover (which, given x, maximises the verifier's acceptance probability) using polynomial $p(|x|)$ space $\to$ PSPACE (and thus $2^{(p|x|)}$ exponential time) $\to$ IP $\subseteq$ PSPACE.

[5] note: the probabilities do not have to sum up to 1!

- Replacing the constant $\frac{3}{4}$ by 1 in the completeness condition does not change the class IP. This is a non-trivial result (the first proof was quite complicated, there are easier proofs now which still require work).
- By contrast, changing $\frac{1}{4}$ to 0 in the soundness condition does not yield IP, but just NP (Variant 1)[6].
- In the IP concept we make use of private random bits → alternative name: notion of private coin interactive proofs. In contrast, there exists the related concept of public random bits (coins) → public coin interactive proofs or Arthur - Merlin proofs

4.2 The Graph Isomorphism Problem, its Complement and Interactive Protocol

Graph isomorphism problem (GI)

Given: 2 undirected graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$.
$n = |V_0| = |V_1|$, $m = |E_0| = |E_1|$.

Question: Are G_0 and G_1 isomorphic ($G_0 \cong sign G_1$)?
$\overline{\text{GI}}$ graph non-isomorqphism

Question: Are G_0 and G_1 not isomorphic? (see Figure 10)

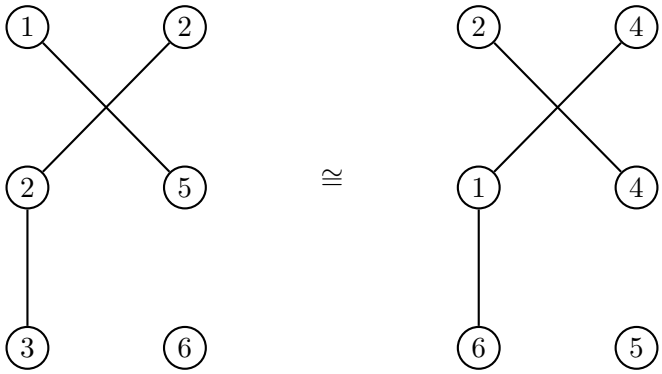

Figure 10: Isomorphic Graphs G_0 and G_1

$$\Pi \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 6 & 3 & 4 & 5 \end{pmatrix}$$

Obviously we have GI ∈ NP and thus GI ∈ IP(1). It is open whether GI is NP-complete (but for certain reasons it is unlikely that GI is NP-complete)

[6] if 0, the prover can never trick the verifier

Question: What about $\overline{\text{GI}}$? Answer: $\overline{\text{GI}} \in$ IP(2)
Interactive protocol for GI
Verifier V: Pick a random bit $i \in 0, 1$. Permute the vertices of G_i randomly by a (random) permutation $\pi \in S_n$ and let $H = \pi(G_i)$. Send H to the prover.
Prover P: Identify which of G_0 or G_1 was used to produce H. Let G_i be such a graph (if $G_0 \cong G_1$ either answer is correct). Send bit $j \in 0, 1$ to V.
Verifier V: Accept if $i = j$ Rejects if $i \neq j$
It is easy to see, that if $G_0 \ncong G_1$ P can always enforce that V accepts. If $G_0 \cong G_1$, then $Prob[Vaccepts] = \frac{1}{2}$, 2 repititions $\leq \frac{1}{4}$
We have just proven that Theorem 4.3 $\overline{\text{GI}} \in$ IP(2)
Comment: This protocol heavily relies on the fact that the random bits of V are private. AM[K] results from IP(K) if the random bits of the verifier are public. Obviously $AM[K] \subseteq IP(K)$. AM is named by Artur Merlin and it is using public random bits. One can show
Theorem (Goldwasser, Sipser) $IP(K) \subseteq AM[k + 2]$ This theorem would suggest an $AM[4]$ protocol for $\overline{\text{GI}}$. There is even an $AM[2]$ protocol for the $\overline{\text{GI}}$.
What's behind this protocol will be dealed with in an exercise (on sheet 3).
Consider there the set S of labeled graphs $S = \{(H, \pi) : H \cong G_0 \text{ or } H \cong G_1\}$ The cardinality of S is different for the 2 cases: $G_0 \cong G_1$ $G_0 \ncong G_1$
Question: Can GI be NP-complete? There is no definite answer, but the following is known: Theorem If GI is NP-complete, then $\Sigma_2 P = \Pi_2 P$. (The polynomial hierarchy caollapses at level 2) Proof: as exercise
Question: How powerful is IP? Of course, $NP \subseteq IP \subseteq PSPACE$. It was conjectured that $IP \subset PSPACE$, but it turned out to be wrong. Answer: Theorem 4.6 (Shamir) $IP = PSPACE$ The idea for the proof will given later in the lecture.

4.3 Interactive protocols with the zero knowledge property(ZKP)

Perfect ZKP and some weaker versions play an important role in applications, in particular in cryptography. prover Bob, Verifier Veronica Informally: Bob does not want to provide Veronica with any knowledge. He just wants to make her accept.(She can not obtain further knowledge on her own) For example, the following problem demonstarates: Given G, does there exist a hamiltonian cycle, Bob does not want to provide the hamiltonian cycle and still convince Veronica that there is a hamiltonian path.
Example: Sketch of room: room 1 and 2 are connected by a secret door, and both have a door into the ante room which has one door to the outside. Bob claims to know the code of the secret door. He wants to convince Veronica that he knows the code without telling or demonstrating it to her. This can be achieved as follows:

1. At the beginning B and V are outside. Then B enters the ante room and closes the door. Bob chooses randomly a bit $i \in 0, 1$ and enters room i and closes the door.

2. Veronica enters the ante room and chooses bit $j \in 0, 1$ and tells it Bob.

3. Bob comes out of one of the 2 rooms.

4. Veronica accepts if Bob comes out of room j.

   Comment: By using the secret door, Bob always chooses the right room, which means he can make Veronica believe he knows the code. If he does not know the code, then there is a just chance of 2. It is easy to see that no information about the code is provided by Bob, thus it is an interactive protocol with the perfect zero knowledge property(PZKP).

Our next goals:

1. provide a concise definition of PZK

2. provide an interactive protocol for GI (Graph isomorphism) with PZK Prop.

More formal: An interactive protocol is said to have the perfect zero knowledge property (PZKP) if the following holds:
Assume that Veronica is not acting in a sincere manner and is executing some arbitrary algorithm V' instead of her standard algorithm V (her goal is to try to "spy out" the secret or parts of the secret)
There is an efficient simulation algorithm which by communicating with Bob achieves the same distribution of messages as the protocol (B, v').

Even more formal:
Let L be a language from NP and let M be a polynomial TM such that $x \in L \Rightarrow \exists y : M(x, y) = 1$ where y is the polynomial certificate of polynomial length.
A pair (P, V) of an interactive protocol (V is a prob. polyn. algorithm) is called a (perfect) zero knowledge proof for L if the following conditions hold:

1. Completeness:

   $\forall x \in L$ and u being a certificate for this fact (i. e. $M(x, u) = 1$): Prob$(out_v(P(x, u), V(x,) = 1) \geq \frac{3}{4}$ where $P(x, u), V(x)$ denotes the interaction of P and V where P gets x and u as input and V gets x as input and $out_v I$ denotes the output (decision) of V at the end of interaction I.

2. Soundness:

   If $x \not\subset L$ then $\forall$ strategy P^* and input u, Prob$(out_V P^*(x, u), V(x) = 1) = \square \leq \frac{1}{4}$ perfect zero knowledge.

   For every prob. polyn. algorithm (statregy) V^*, there exists a prob. algorithm S^* (a stand-alone algorithm), whose expected running time is polynomial, such that $\forall x \in L$ and u being a certificate for $x \in L$ we have $out_{V^*}(P(x, u), V^*(x)) \equiv S^*(x)$ (both sides are random variables, with identical distribution)

   I.e. these two random variables are identically distributed, even though S^* does not have access to a certificate for x.

   The algorithm S^* is called simulator for V^* as it simulates the outcome of V^*'s interaction with the prover.

Zero knowledge property means that the verifier cannot learn anything new, not even by deviating from the standard protocol algorithm V and using V^* instead. (The same as from the interaction with P can be learnt by just applying S^*.)

Comments:

- It does not influence the resulting concept that we ask only for expected polynomial running time for S^*.
- Analogous definition can be set up for languages from sets $\neq$ NP.

Now: PZK as class of languages for which there exists an interactive protocol (proof systems) with PZK property.

Theorem 4.3. GI $\in$ PZK

Proof. We provide a PZK interactive protocol for GI Graphs G_0, G_1 are known to both prover P and verifier V.
If $G_0 \cong G_1$, then P further has access to a permutation $\Pi \in S_n$ such that $G_1 = \Pi(G_0)$.

1. Prover P chooses random permutation $\Pi' \in S_n$ and sends the graph $H = \Pi'(G_1)$ to the verifier. (e.g. by sending adjacency lists)
2. Verifier chooses random bit $b \in \{0, 1\}$ and sends it to the prover.
3. The prover sends a permutation $\tilde{\Pi} \in S_n$ to the verifier.
   - If $b = 1$, the prover chooses $\tilde{\Pi} = \Pi'$
   - If $b = 0$, the prover chooses $\tilde{\Pi} = \Pi' \circ \Pi$ $(\tilde{\Pi}(G_0) = \Pi' \circ \Pi(G_0) = \Pi'(G_0) = H)$
4. The verifier accepts iff $H = \tilde{\pi}(G_b)$.

If both prover and verifier follow this protocol, then the verifier will accept if $G_0 \cong G_1$ with probability 1 ($\rightarrow$ Completeness).
For soundness: suppose $G_0 \ncong G_1$. Then the verifier will reject with a probability $\geq \frac{1}{2}$ (can be made larger by repetitions).
(Regardless of the prover's strategy, the graph H which is sent in the first message cannot be isomorphic to both G_0 and G_1 and so there exists a $b^* \in \{0, 1\}$ such that $H \ncong G_{b^*}$. The verifier will choose $b = b^*$ with probability $\frac{1}{2}$ and then the prover i not able to find a permutation $\tilde{\Pi}$ such that $H = \tilde{\Pi}(G_b)$) $\square$

Definition (Zero Knowledge): Let V^* be an arbitrary verifier strategy. Consider the following simulation algorithm S^*: On input G_0, G_1, S^* chooses bit $b' \in \{0, 1\}$ and a random permutation $\Pi' \in S_n$ and computes $H = \Pi'(G_b)$.
Interesting case: $G_0 \cong G_1$ (otherwise there is no secret) $\rightarrow$ assume this.
Then S^* feeds H into V^* to obtain a bit $b \in \{0, 1\}$. If $b = b'$, then S^* sends Π' to V^* and outputs whatever V^* outputs.
If $b \neq b'$, then S^* restarts.

Question: How are the messages of S^* distributed?

Answer: They are distributed in exactly the same way as the prover's first message (a random graph that is isomorphic to both G_0 and G_1.
This implies that H reveals nothing about the choice of b', hence the probability that $b = b'$ is $\frac{1}{2}$.
In this case ($b = b'$), the messages H and Π' that V^* sees are distributed in exactly the same way that it gets in the real interaction with prover.
Because S^* succeeds in getting $b = b'$ with probability $\frac{1}{2}$, the probability that it needs k iterations is Z^{-k}.
$\rightarrow$ the expected running time of S^* is $T(n) \cdot \sum_{k=1}^{\infty} 2^{-k} = O(T(n)) \rightarrow S^*$ has expected polynomial running time. ($T(n)$ is the running time of V^*).

4.4 Weaker Zero Knowledge Property

We have now seen that there exist languages $L \in$ NP such that $L \in$ PZK.

Question: Do there exist PZK protocols for NP-complete problems? E.g. for Hamiltonian path/cycle?

Answer: Probably no, since the following can be shown: PZK $\subseteq$ IP(2) $\cap$ Co-IP(2)[7].
The existence of a PZK protocol for an NP-complete problem would imply Σ_2P= Π_2P.
$\rightarrow$ Let's try to come up with a weaker concept of zero knowledge than the perfect zero knowledge property.

4.4.1 First attempt

Statistical zero knowledge property (SZK): here we only ask that the distribution generated by S^* and the one generated by the real protocol differ only in a neglectibally small manner ($\varepsilon(n)$ is negl. small if $\varepsilon(n)$ is superpolyn. small, i. e. $\exists$ polyn. p such that $\varepsilon(n) \leq \frac{1}{p(n)}$ for all sufficiently large n).
Unfortunately, the same result as for PZK holds for SZK with respect to NP-completeness.

4.5 Computational Zero Knowledge Property (CZK)

The kind of CZK protocols for NP-complete problems exist under the assumption that one way functions exist (this is a stronger assumption than assuming P $\neq$ NP).
It is an open problem to decide whether the the assumption P $\neq$ NP suffices. If P = NP, then such protocol cannot exist. $\rightarrow$ relationship to concepts of (perfect) secrecy in cryptography.

[7] for GI: GI $\in$ IP(1) $\in$ NP. $\overline{\text{GI}} \in$ IP(2) $\rightarrow$ GI $\in$ Co-IP

4.5.1 Hamiltonian Circuit

Theorem 4.4. There exists a CZK protocol for the Hamiltonian circuit problem HC.

Proof. Consider the following interactive protocol:
Let $G = (V, E), |V| = n$ be the given undirected graph. Henceforth we assume that we have c (correct) bit commitment method.

1. Bob chooses a random permutation $\Pi \in S_n$. Bob determines $\Pi(G)$ (e.g. as edge lists).

   Bob sends bit commitments for Π and $\Pi(G)$.

2. Veronica chooses randomly a bit, $i \in \{0, 1\}$ and sends it to Bob.

3. If $i = 0$, Bob has to uncover the bits for Π and $\Pi(G)$.

   If $i = 1$, Bob uncovers the bits of n edges of $\Pi(G)$ ($\equiv n$ edges of G) such that if there exists a Hamiltonian circuit in G the uncovered edges form a Hamiltonian circuit. Otherwise he chooses the edges arbitrarily.

4. If $i = 0$, Veronica accepts if Bob has not cheated in step 3, i.e. he uncovered a proper permutation Π' and the corresponding $\Pi'(G)$, otherwise she rejects. If $i = 1$, Veronica accepts if Bob uncovered the edges of a Ham. circuit.

Prove that this is an IP

- If $G \in$ HC (i.e. G contains a Ham. circ.), Bob can always make Veronica accept. Error prob. 0
- If $G \notin$ HC, then Bob cannot pass the two tests for $i = 0$ and $i = 1$ at the same time, if he permutes the correct graph, he will pass the $i = 0$ test, but fail for $i = 1$. If he cheats with $\Pi, \Pi(G)$, he can pass the test, but fails for the $i = 0$ test. $\rightarrow$ error probability $\frac{1}{2}$ $\rightarrow$ repeat 2 times $\rightarrow$ error prob. $\frac{1}{4}$.

Now: focus on the CZK property.
Consider a protocol (B, V') and the following simulation algorithm S for Bob.

1. Choose randomly a bit $i' \in \{0, 1\}$ and continue with the hypothesis that the V' algorithm chooses $i = i'$. If $i' = 0$, send bit commit
2. Simulate V' for the data sent by Bob.
3. If $i \neq i'$, restart
4. Simulate the action of Bob in step 3 of the interactive protocol
   - $\mathbb{P}[i = i'] = \frac{1}{2}$ $\rightarrow$ expected number of iterations in constant $\rightarrow$ polyn. time

- The distribution of the messages sent by the simulation algorithm S and by the protocol (B, V') should be distinguishable within polynomial time only with neglectibly small probability

Formal proof: exercise

□

4.5.2 3-Colourability

Another example for an interactive protocol for an NP-complete problem with CZK property

Given: undirected graph $G = (V, E)$, $|V| = n$

Task: find a 3-colouring of G: $\Phi : V \to \{1, 2, 3\}$ such that $\{i, j\} \in E \Rightarrow \Phi(i) \neq \Phi(j)$
Consider the following protocol:

1. Bob chooses randomly a permutation $\tau \in S_3$ (permutation of colour set). For $i = 1$ to n, Bob sends bit commitments for $\tau(\Phi(i))$

   Comment: if Bob knows a 3-colouring, he chooses Phi as an arbitrary $\Phi : V \to \{1, 2, 3\}$

2. Veronica chooses randomly an edge $e(= \{i, j\}) \in E$ and sends it to Bob.

3. Bob needs to uncover the values sent for i and j.

4. Veronica checks whether the two colours assigned to i and j are different and accepts if this is the case and rejects otherwise

Again, if the $G \in$ 3COL, then Bob will always make Veronica accept.
If $G \notin$ 3COL, Veronica catches a cheating Bob with a small probability if the protocol is used only once.
It can be shown that by repeating this process $r \times |E|$ times, the error probability is around $e^{-r} \to 1 - e^{-r}$ (easy exercise).

Comment on a possible implementation of the above protocol: binary codes for colours 00, 01, 11, consider colorings of form $V \to \{00, 01, 11\}$.

Suppose Bob uses an RSA-based bit commitment method. Bob chooses randomly the permutation τ of the colour set. Bob generates $|V|$ RSA public-private key pairs (p_i, q_i, d_i, e_i), one for each vertex $i \in V$
For each vertex i, Bob computes the encoding (y_i, y_i') accordingly.
Let $b_i b_i'$ bet the two bits of $\tau(\Phi(i))$ (the colour of i permutations under τ).
Then $y_i = (2x_i + b_i)^{e_i} \mod p_i q_i$, $y_i' = (2x_i' + b_i')^{e_i} \mod p_i q_i$, where x_i and x_i' are random integer numbers greater than $\frac{p_i q_i}{2}$ (private computation of Bob)
Bob sends $(e_i, p_i q_i, y, y_i')$ (implementation of bit commitment) for each $i \in V$ to Veronica.

Veronica randomly chooses an edge $\{i, j\} \in E$ and asks Bob to uncover the sent information for i and j. Bob sends Veronica the secret keys d_i and d_j for the end points of the selected edge.
$\rightarrow$ This allows Veronica to compute $b_i = (y_i^{d_i} \mod p_i q_i) \mod 2$ and similarly for b_i', b_j, b_j'.
Now just check whether $b_i b_i' \neq b_j b_j'$.
Again, it is not hard to show that Veronica is not learning anything about the 3COL from the interactive protocol (CZK property).

4.6 Multiple Provers

If we consider several independent provers, one obtains the concept of multi-prover interactive protocols (systems) $\rightarrow$ complexity class mIP
It suffices to consider 2 independent provers.
It was proven by Babai, Fortnow and Lund (1990) that mIP = NEXP (very powerful!).
Unless NEXP = PSPACE (= IP), mIP is more powerful than IP.

5 Probabilistically Checkable Proofs (PCP) and the PCP-Theorem

Definition: Let $r, q : \mathbb{N} \to \mathbb{N}$. An $(r(n), q(n))$[8] PCP verifier is an algorithm V with polynomial running time and the following properties:

- For an input x of length n and a prof $B \in \{0,1\}^*$, the algorithm V has access to x and to a random vector $r \in \{0,1\}^{r(n)}$ (V has access to $r(n)$ random bits).
- With this information, V computes up to $O(q(n))$ positions and then is provided with the corresponding bits of the proof.

  Finally, V has to decide whether or not to accept the proof. $V(x, r, B) \in \{0, 1\}$ (0 for reject, 1 for accept).

Remark: Since V runs in polynomial time, only polynomials for $r(n)$ and $q(n)$ make sense.

Definition: A decision problem L belongs to the class PCP$(r(n), q(n))$ if there exists an $(r(n), q(n))$ verifier V such that

- Completeness: $\forall x \in L \; \exists$ proof B such that $\mathbb{P}_z(V(x, z, B) = 1) = 1$
- Soundness: $\forall x \notin L \; \forall$ proofs B' we have $\mathbb{P}_z(V(x, z, B') = 1) \leq \frac{1}{2}$.

[8] resource bounds

Simple observations:

- $\mathrm{PCP}(0, \log(n)) \longrightarrow \mathrm{P}$
- $\mathrm{PCP}(0, \mathrm{poly}(n)) \longrightarrow \mathrm{NP}$

The PCP Theorem states that $\mathrm{NP} = \mathrm{PCP}(\log n, 1)$. Here: proof for weaker assumption, allowing n^3 random bits.

Theorem 5.1. $L \in \mathrm{PCP}(r(n), q(n)) \Rightarrow \exists$ non-deterministic algorithm which decides L in $2^{O(r(n)+\log n}$ time.

Proof. Let pol. $p(n)$ be the running time of the PCP verifier V.
$\rightarrow$ at most $p(n) \cdot 2^{O(r(n)}$ bits are read. Our non-deterministic algorithm guesses these bits.
$\rightarrow$ random bit string
We simulate the run of V for each of the strings and accept if all simulations end with acceptance.
$\rightarrow$ running time is as claimed. □

Consequence: We get NP as $\mathrm{PCP}(\log n, \mathrm{poly}(n)) = \cup$ PCP $(\log n, n^k)$, $k \geq 0$.

Theorem 5.2. (PCP-Theorem; Arora, Lund, Motwani 1992) $\mathrm{NP} = \mathrm{PCP}(\log(n), 1)$

Ideas of a proof: find a $(n^3, 1) - PCP$ verifier for 3SAT

A proof verifier will consist of the following four components

- linearity test
- robust function evaluation
- consistency test
- verifier for nice proofs

Verifier for nice proofs

1. Arithmetisation

   Let 3SAT instance with variables $x_1, \ldots, x_n$, clauses $c_1, \ldots, c_m$, 3SAT formula Φ be given.

Boolean form Φ	Arithmetisation
x_i	$1 - x_i$
$\overline{x_i}$	x_i
$\wedge$	$+$
$\vee$	$\cdot$
true	1
false	0

Example:

$$\underbrace{(x_1 \vee \overline{x_2} \vee x_3)} \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

$$(1 - x_1) \cdot x_2(1 - x_3) + x_1(1 - x_2)(1 - x_4)$$

Observation: Arithmetisation leads to a polynomial with degree ≤ 3. Let's call the resulting polynomial $P(x_1, \ldots, x_n)$.

- Let $a \in \{0,1\}^n$. $P(a) = 0 \Leftrightarrow a$ is a satisfiable truth assignment.
- More generally, $P(a)$ delivers the number of not satisfied clauses.

2. Switch to mod 2 arithmetics $\rightarrow$ the coefficients of $P \mod 2$ are 0 or 1 $\rightarrow$ indicator functions.

   Drawback $P(a) \cong 1 \mod 2 \Rightarrow a$ is not satisfied (no mistake possible, but if $P(a) \cong 0 \mod 2$, we do not know

   $\rightarrow$ make use of randomness

3. Consider only a randomly selected subset of clauses.

   Let S be a random 0-1 vector of dimension m. Then let $P^{(\phi)}$ be the sum of all clause polynomials P_i (term for clause c_i) where $\phi_i = 1$

   $P^{(\phi)}(a) = \sum_{i=1}^{m} \phi_i P_i(a)$

   We have $P^{(\phi)}(a) = 0$ if a is satisfying.

   $$\mathbb{P}(P^{(\phi)}(a) \cong 0(2)) = \mathbb{P}(P^{(\phi)}(a) \cong 1(2)) = 1(2) = \frac{1}{2}$$

   if a is not satisfying.

   Problem: a $(\gamma, 1)$ PCP verifier can look at only a constant number of proof bits and so we cannot use $a \in \{0,1\}^n$ as certificate. We need a method to compute $P(a)[P^{(\phi)}(a)] \mod 2$ without knowing a.

4. Let Ψ be a polynomial of degree ≤ 3 over $\mathbb{Z}_2$. Ψ consists (in the explicit for m after multiplying out) of terms of the following type:

   $\gamma_\Psi \in \{0,1\}$ constant term

   x_i $i \in I_\Psi^1$ (index set)

   $x_i \cdot x_j$ $(i,j) \in I_\Psi^2$

   $x_i \cdot x_j \cdot x_k$ $(i,j,k) \in I_\Psi^3$

   Define the linear function L_1^a, L_2^a, L_3^a

   - $L_1^a : \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2$, $L_1^a(y_1, \ldots, y_n) = \sum_{i=1}^{n} a_i \cdot y_i$

- $L_2^a : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2$, $L_2^a(y_1, \ldots, y_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i \cdot a_j \cdot y_{ij}$
- $L_1^a : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2$, $L_3^a(y_1, \ldots, y_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} a_i \cdot a_j \cdot a_k \cdot y_{ijk}$

The function table for L_1^a has size 2^n, for L_2^a has size 2^{n^2}, for L_3^a has size 2^{n^3}. together $2^n + 2^{n^2} + 2^{n^3}$

Let $\gamma_\Psi^1, \gamma_\Psi^2, \gamma_\Psi^3$ be the indicator functions for the sets $I_\Psi^1, I_\Psi^2, I_\Psi^3$ (γ_Ψ^2 has a 1 in position $(i,j) \Leftrightarrow (i,j) \in I_\Psi^2$, similarly for γ_Ψ^1, γ_Ψ^3)

We have that

$$\Psi(a) = \gamma_\Psi + \underbrace{L_1^a(\gamma_4^1)} + \underbrace{L_2^a(\gamma_4^2)} + \underbrace{L_3^a(\gamma_4 31)}$$

$\to V$ is able to evaluate $P(a)$ or $P^\phi(a) \mod 2$ without the help of the three underbraced values above and without having access to a.

What we have so far proved the concept for a well-formed ("nice") proof.

Function tables L_1^a, L_2^a, L_3^a. Check $P^\Psi(a)$ for a set of ranom ϕs and accept if they all lead to the result 0 and reject if not.

The number of required random bits is $O(n^3)$ since the number of clauses on a non-trivial 3SAT instance $\leq 2n + 4\binom{n}{2} + 8\binom{n}{3} = O(n^3)$.

Now: deal with the situation that "not everything is behaving nicely".
We can assume that the proofs are of the right length.

Linearity test Definition: $f : \mathbb{Z}_2^m \to \mathbb{Z}_2$ is linear if $f(x+y) = f(x) + f(y) \forall x, y \in \mathbb{Z}_2^m$.
f is called δ-close to $g(f, g : \mathbb{Z}_2^m \to \mathbb{Z}_2)$ if $\mathbb{P}_x(f(x) \neq g(x)) \leq \delta$ (x chosen with respect to uniform distribution over $\mathbb{Z}_2^m$).
We call a function f "almost linear" if it is a δ-close function to a linear function for a suitable $\delta > 0$ (see later).
Idea for linear function: Choose randomly and independently $x, y \in \mathbb{Z}_2^m$ and call f non-linear if $f(x) + f(y) \neq f(x+y)$.
Properties

1. If f is linear, f always passes the test
2. If f is δ-close to a linear function for a $\delta < \frac{1}{3}$, then f passes the test with probability $1 - \frac{\delta}{2}$

Proof:

1. obvious

2. we need to show that $\mathbb{P}_{x,y}(f(x+y) \neq f(x) + f(y)) \neq \frac{\delta}{2}$ $(*)$

   We will work with $(*)$.

   Constructive approach: construct a linear function g which is δ-close to f.

   Let $a \in \mathbb{Z}_2^m$. We define $g(a)$ as follows: We compute $f(a+b) - f(b)$ for all $b \in \mathbb{Z}_2^m$. We set $g(a) = 0$ if 0 appears more often or just as often as 1 and set $\phi(a) = 1$ otherwise.

   To show that ϕ is linear and δ-close to f.

   - Suppose that f and g are not δ-close $\Rightarrow \mathbb{P}_x(f(x) \leq g(x)) > \delta$. Due to the construction of g, we have $\mathbb{P}_x(g(a) = f(a+y) - f(y)) \geq \frac{1}{2}$.
   - $\mathbb{P}_{x,y}(f(x+y) - f(y) \leq f(x)) \geq \mathbb{P}_{x,y}(f(x+y) - f(y) = g(x),\ g(x) \neq f(x)) = \frac{1}{2^m} \sum\limits_{a \in \mathbb{Z}_2^m} P_y(f(a+y) - f(y) = g(a), g(a) \neq f(a))$

     We have $\forall a \in \mathbb{Z}_2^m f(a) = g(a)$ or $f(a) \neq g(a)$.

     If $f(a) = g(a)$, then the above probability is 0 and in the second case $f(a) \neq g(a)$ can be omitted.

     $\Rightarrow P_{x,y}(f(x+y) - f(y) \neq f(x)) \geq \frac{1}{2^m} \sum\limits_{a \in \mathbb{Z}_2^m} P_y(f(a+y) - f(y) = g(a)) \geq \frac{1}{2^m} \sum\limits_{a \in \mathbb{Z}_2^m} \delta.$

     Last step: follows because from $P_x(f(x) \neq g(x)) > \delta$ we get that, for more than $2^m\delta$ of all a's $\in \mathbb{Z}_2^m$ we have $g(a) \neq f(a)$.

     $\rightarrow$ leads to contradiction and leads to the claimed result.
   - Show that g is linear

     Consider $p(a) := \mathbb{P}_x(g(a) = f(x+a) - f(x)$. Obviously $p(a) \geq \frac{1}{2}$ (due to the choice of g). We will show that $p(a) \geq 1 - \delta$.

     Let $x \in \mathbb{Z}_2^m$ randomly chosen $\Rightarrow x + a \in \mathbb{Z}_2^m$ randomly chosen.

     Event 1: $\mathbb{P}_{x,y}(f(x+a)+f(y) \neq f(x+a+y)) \leq \frac{\delta}{2}$. (Assumption $(*)$, equivalent to (P2), compare the arguments from the last lecture)

     Analogously we have event 2: $\mathbb{P}_{x,y}(f(x) + f(y+a) \neq f(x+a+y)) \leq \frac{\delta}{2}$

     $\Rightarrow$ Probability of the union of the two events is bounded from above by δ and the probability of the complement is bounded from below by $1 - \delta$.

     According to de Morgan, this complement is the intersection of the events $f(x) + f(y+a) = f(x+y+a)$, $f(y) + f(x+a) = f(x+y+a)$

     $\rightarrow$ subset of the event $f(x) + f(y+a) = f(y) + f(x+a)$

     $\Rightarrow \mathbb{P}_{x,y}(f(x+a) + f(y) = f(y+a) + f(x)) \geq 1 - \delta \Leftrightarrow \mathbb{P}_{x,y}(f(x+a) - f(x) = f(y+a) - f(y)) \geq 1 - \delta$ $(**)$

     $(**) = \sum\limits_z \in \{0,1\} \mathbb{P}_{x,y}(f(x+a) - f(x) = z, f(y+a) - f(y) = z) = \sum\limits_z \in \{0,1\} \mathbb{P}_x(f(x+a) - f(x) = z) \cdot \mathbb{P}_y(f(y+a) - f(y) = z) = \sum\limits_z \in \{0,1\} \mathbb{P}_x(f(x+a) - f(x) = z)^2$

Now observe

For

- $z = g(a)\mathbb{P}_y(f(x+a) - f(x) \geq z) = p(a)$
- $z \leq g(a)\mathbb{P}_y(f(x+a) - f(x) \geq z) = 1 - p(a)$

Rewriting the above, we get $1 - \delta \leq [p(a)]^2 + (1 - p(a))^2$

We know that $p(a) > \frac{1}{2} \rightarrow 1 - p(a) \leq \frac{1}{2} \leq p(a)$

So we have $(p(a))^2 + (1 - p(a))^2 \leq \underbrace{(p(a))^2}_{p(a)} + p(a)(1 - p(a))$

$\Rightarrow p(a) \geq 1 - \delta$ as we wanted to prove.

Event I: $p(a) = \mathbb{P}_x(g(a) = f(a + x) - f(x)) \geq 1 - \delta$

Event II: $p(b) = \mathbb{P}_x(g(a) = f(b+a+y) - f(a+y)) \geq 1 - \delta$ (remark if x is uniformly distributed, this is true for $a + x$ as well)

Event III: $p(a + b) = \mathbb{P}_x(g(a + b) = f(a + b + x) - f(x)) \geq 1 - \delta$

$\rightarrow$ probability for the intersection of all 3 events is $\geq 1 - 3\delta$.

Add I + II and subtract II from the sum

$\Rightarrow \mathbb{P}_x(g(a) + g(b) = g(a + b)) \geq 1 - 3\delta$.

Since $\delta < \frac{1}{3}$ we obtained that $\mathbb{P}_x(g(a) + g(b) = g(a + b)) > 0$.

since the event $g(a) + g(b) = g(a + b)$ does not depend on x, this probability must be 1 and thus g is linear!

Robust function evaluator Goal: for $\delta < \frac{1}{3}$, we want the following properties fulfilled:

1. If f is linear, the function evaluator should deliver $f(a)$ as value for $a \forall a \in \mathbb{Z}_2^m$
2. If f is δ-close to linear function g, then the robust function evaluator should provide $g(a)$ as value with an error probability bounded by 2δ.

To determine the value the function evaluator returns for $f(a)$, we proceed as follows: Choose $x \in \mathbb{Z}_2^m$ randomly and compute $f(x + a) - f(x)$ and returns this as a result for the value $f(a)$

1. is trivially fulfilled.
2. f is δ-close to a linear function $g \Rightarrow \mathbb{P}_x(f(x) = g(x) \geq 1 - \delta \rightarrow \mathbb{P}_x(f(x + a) = g(x + a)) \geq 1 - \delta$

   $\rightarrow$ probability the both events occur is $\geq 1 - 2\delta$.

   If both events hold, we get $f(x + a) - f(x) = f(x + a) - g(x) \overset{g \text{ is linear}}{=} g(a)$

Consistency test We have given 3 function tables for f_1, f_2, f_3 (in our case L_1^a, L_2^a, L_3^a). f_1, f_2, f_3 are either linear or δ-close to linear functions g_1, g_2, g_3 (otherwise they fail the linearity test).
Here we assume $\delta \leq \frac{1}{24}$. Following properties wanted:

1. If the function tables for f_1, f_2, f_3 represent linear function of the type L_1^a, L_2^a, L_3^a) (for $a \in \{0,1\}^n$), then the consistency should always be passed.

2. If there does not exist $a \in \{0,1\}^n$ such that the function tables for f_1, f_2, f_3 are δ-close to L_1^a, L_2^a, L_3^a), then the consistency test should be passed only with a probability bounded by an appropriately chosen constant (bounded error probability)

Choose randomly and independently $x, x', x'' \in \mathbb{Z}_2^n, y \in \mathbb{Z}_2^n$.
Define

- $x \circ x'$ by $(x \circ x')_{ij} = x_i \cdot x'_j \quad \rightarrow \dim n^2$
- $x'' \circ y$ by $(x'' \circ y)_{ijk} = x''_i \cdot y_{jk} \quad \rightarrow \dim n^3$

Use robust function evaluator to get the values

- b for $f_1(x)$
- b' for $f_1(x')$
- b'' for $f_1(x'')$
- c for $f_2(x \circ x')$
- c' for $f_2(y)$
- d for $f_3(x'' \circ y)$

Consistency test is passed if $b \cdot b' = c$ and $b'' \cdot c' = d$
L_1^a, L_2^a, L_3^a) pass this test because $L_1^a(x) \cdot L_1^a(x') \cdot L_1^a(x'') = L_2^a(x \circ x')$ and $L_1^a(x'') \cdot L_2^a(y) = L_3^a(x'' \circ y)$
Proof for second property: since $\delta < \frac{1}{24} \rightarrow 2\delta < \frac{1}{12}$. Since we perform 6 evaluations, each with error probability $< \frac{1}{12}$, the overall error probability for evaluation part is $< \frac{1}{2}$.
Now assume that the function evaluations are correct, but the function tables are inconsistent. Consider here the $bb' \neq c$ case (the $b''c' \neq d$ case works similarly).
Consider x and x' as column vectors. Idea of consistency test is comparing $x^t Ax'$ and $x^t Bx'$ and exploiting that if $A \neq B$ for random x and x' we get $x^t Ax' \neq x^t Bx'$ with probability $\geq \frac{1}{4}$.

Combination of all together leads to the claimed result.

Closing remark It can be shown that PCP(poly(n), 1) = NEXP.
(Idea of proof: combination of the proof for the PCP theorem and the proof of theorem of Shamir that IP = PSPACE)

Part II
Approximation from a Complexity Point of View

6 Introduction

Motivation behind looking for approximative solutions is that many optimisation problems are very hard.
We need a measure to evaluate the quality of an approximate solution.

Example 1

Instance	optimal value	approximate solution value
1	17	27
2	1700	1720

Typically, one works with a relative error message (independent of scaling) and in the example this then favours the situation in instance 2.
In the following, we consider optimisation problems where x denotes the input. $S(x)$ denotes the set of feasible solutions and $v(x, s)$ the objective function value of a feasible solution $s \in S(x)$.
Assumptions:

- $v(x, s) > 0 \forall x, s$: simplification to make the following definitions easier.
- $\forall x, s \in S(x)$ the encoding length of s and $v(x, s)$ are polynomials in encoding length of x. Background: we want approximation algorithms to run in polynomial time.

7 Definitions

7.1 Approximation Ratio

$r(x, s) := \frac{v(x,s)}{v_{opt}(x)}$ where $v_{opt}(x)$ is the optimal value for instance with input x and $s \in S(x)$ for minimisation problem.
$r(x, y) := \frac{v_{opt}(x)}{v(x,s)}$ where $s \in S(x)$ for maximisation problem.

Remarks

- This definition is possible because we assumed $v(x, s) > 0$.
- We always get $r(x, s) \geq 1$. The closer we are to 1, the better is s.
- If $r(x, s) \leq c$ with $c \geq 1$, we call s a c-approximation.

Worst case p.o.w: For an algorithm A let $s_A(x) \in S(x)$ be the solution returned by algorithm A.
Approximation quality (ratio) of A: $r_A(x) := r(x, S_A(x))$.
For algorithm A we define $r_A(n) := \max\{r_A(x) \mid \underbrace{|x|}_{\text{encoding length}} \leq n\}$.
If $r_A(n) \leq c$ for a $c \geq 1$ and all $n \in \mathbb{N}$ we call A a c-approximation or c-factor approximation.
But sometimes $r_A(n)$ is not the most suitable quality measure.
Let's consider the bin packing problem (BP) and the so-called BFD (best fit decreasing) algorithm for BP.

Given: numbers (item sizes) $a, \ldots, a_n \in (0, 1)$.

Goal: pack the items in as few bins (of unit capacity) as possible.
One can show that $v(x, s_{BFD}(x) \leq \frac{11}{9} \cdot v_{opt}(x) + 4$.
$r_{BFD}(x) \leq \frac{11}{9} + \frac{4}{v_{opt}}$.
Now let us assume that $v_opt(x) \geq 2 \rightarrow r_{BFD}(x) \leq \frac{29}{9}$.
For large instances, $\frac{29}{9}$ is far from the truth. For $v_{opt} \rightarrow \infty$, the ratio much rather behaves like $\frac{11}{9}$.
This motivates the following definition

7.2 Adjusted Approximation Ratio

$$r_A^\infty := inf\{b \left| \begin{array}{l} \forall \varepsilon > 0 \exists v(\varepsilon) > 0 \\ \forall x, v_{opt}(x) \geq v(\varepsilon) : r_A(x) \leq b + \varepsilon \end{array} \right.$$

It is easy to see that $r_A BFD^\infty = \frac{11}{9}$ (this is better than the best approximation algorithm for BP can get in terms of $r_A(n)$ unless P = NP)

8 Complexity Classes

8.1 Complexity classes APX, APX*

Definition: Let $r(n) : \mathbb{N} \rightarrow [0, \infty)$ with $r(n + 1) \geq r(n) \forall n \in \mathbb{N}$.
The complexity class APX($r(n)$) contains all optimisation problems which can be solved by an approximation algorithm A with (max.) approximate ratio $r_A(n) \leq r(n)$ where A is required to run in polynomial time.

$\text{APX} := \bigcup_{c \geq 1,\ \text{const}} \text{APX}(c)$.
(problems which are approximated with a constant factor in polynomial time)

$\text{APX}^* := \bigcap_{c > 1,\ \text{const}} \text{APX}(c)$.
(problems which are approximations in polynomial time with approximation factor arbitrarily close to 1)

8.2 PTAS

Definition: PTAS (Polynomial Time Approximation Scheme: A PTAS for an optimisation problem Q is an algorithm, which for an input (x, ε) where x is an input for Q and $\varepsilon > 0$ is a rational number provides a $(1+\varepsilon)$-optimal solution (approximation factor $\leq 1 + \varepsilon$) for Q in time polynomial in $|x|$.
PTAS as a complexity class class: contains all optimisation problems for which there exists a PTAS.

8.3 FPTAS

Definition: FPTAS (Fully Polynomial Time Approximation Scheme: see PTAS, but the running time is required to be polynomial in $|x|$ and $\frac{1}{\varepsilon}$.

Comment: The running time of a PTAS can be exponential in $\frac{1}{\varepsilon}$, e.g. $O(n^{\frac{1}{\varepsilon}})$, while for a FPTAS this is forbidden.

8.4 Relation

Obviously, we have P $\subseteq$ FPTAS $\subseteq$ PTAS $\subseteq$ APX.

9 Examples

9.1 MAX CLIQUE

Given: undirected graph $G = (V, E)$, $|V| = n$.

Goal: find a clique with maximum cardinality (complete subgraph).

Algorithms:

1. Trivial approximation alg.: output an arbitrary single vertex $\rightarrow$ ratio $\leq n$
2. Choose $k \in \mathbb{N}, k < n$. Consider all subsets of V with cardinality k and output best one $\rightarrow$ ratio $\leq \frac{n}{k}$.

Later we will see that MAXCLIQUE $\notin$ APX.

9.2 VERTEX COVER

Given: undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$.

Goal: Find vertex cover $V' \subseteq V$ such that each edge in e is covered by a vertex in V' (each edge has at least end point in V') and such that V' is of minimum cardinality. NP-complete for general graphs.

Let's consider the following approximation algorithm:

1. $E' = \varnothing$
2. As long as there still exists an edge $\{u, v\}$ such that neither u nor v are covered by an edge in E', choose such an edge and insert it into E'.
3. r' s the set of end vertices of the edges in E'.

Claim 1: E' is a maximal matching (not necessarily a matching of maximum cardinality).

Proof. E' is a matching due to the construction that only edges $\{u, v\}$ which are yet uncovered are added. E' is maximal because otherwise a new edge could be added to E'. □

Claim 2: V' is a vertex cover.

Proof. Suppose V' is not a vertex cover $\Rightarrow$ $\exists$ edge $\{u, v\}$ which is not covered by $V' \Rightarrow u, v \notin V'$.
$\{u, v\} \notin E'$, that is, we could add $\{u, v\}$ to E' and so contradict the maximality. □

Claim 2: the algorithm above is a 2-approximation algorithm for VERTEX COVER.

Proof. Suppose $|E'| = k \Rightarrow |V'| \leq 2K$. The claim now follows immediately □

$\rightarrow$ VERTEX COVER $\in$ APX.

Remark: There exist alternative approaches which also lead to 2-approximation for VERTEX COVER.

9.3 MAX 3-SAT

Given: 3SAT formula Φ in variables $x_1, \ldots, x_n$ with clauses $c_1, \ldots, c_n$, exactly 3 literals per clause.

Task: Find a truth assignment that satisfies as many clauses as possible.
MAX 3-SAT is NP-hard.
Consider the following simple randomised algorithm:

- Independently assign 0-1 (true-false) values to the variables $x_1, \ldots, x_n$ according the the uniform distribution.
  $\rightarrow X_j = \begin{cases} 1 & \text{if clause } c_j \text{ is satisfied by truth assignment } x \\ 0 & \text{otherwise} \end{cases}$
- We want to maximise $X = \sum x_j$ or rather the expected value $E(X)$.
- $E(X) = \sum_{j=1}^{m} E(X_j) = \frac{7}{8}m$, since in 7 out of 8 cases, c_j is fulfilled
  $\rightarrow$ Randomised algorithm of quality $\frac{7}{8}$.

Next goal: deterministic $\frac{8}{7}$-approximation algorithm.
Use a derandomisation method based on conditional probabilities: fix the values for $x_1, \ldots x_n$ sequentially according to a suitably chosen order.
We need to guarantee that at each step of this fixing process, we are left with a situation s.t. for a (suitable) random choice of the remaining variables, $\geq 78m$ (or expected value) clauses can be satisfied.
During the course of the algorithm, we end up with 0 to 3 fixed literals.
Let $b \in \{0, 1\}$.
Consider $a_j := E(X_j | x_n = b)$, $j = 1, \ldots, m$.
If by setting $x_n = b$ the clause c_j is already satisfied, then we get $a_j = 1$. Otherwise $\exists k \in \{1, 2, 3\}$ not yet fixed literals in c_j.
Then it is easy to see that $a_j = \frac{2^k - 1}{2^k} = 1 - \frac{1}{2^k}$.
We set $a_j = 0$ if the clause c_j is already fixed to be non-satisfied.
$E(X | x_n = b) = \sum_{j=1}^{m} E(x_j | x_n = b) = \sum_{j=1}^{m} a_j$ is computable in $O(m)$ time.
$E(X) = \frac{1}{2}E(X | x_n = 1) + \frac{1}{2}E(X | x_n = 0)$.
We know that $E(X) \geq \frac{7}{8} \Rightarrow \exists$ a choice for $x_n = b, b \in \{0, 1\}$ such that $E(X | x_n = b) \geq \frac{7}{8}m$.
Repeat this iteratively until all variables are fixed. Running time $O(n \cdot m)$, approximation ratio $\frac{7}{8}$.

9.4 Simple PTAS Problem

Consider the following 2 machine scheduling problem:

Given: set $J = \{1, \ldots, n\}$ jobs, 2 identical machines, p_j processing time for job j, with $j = 1, \ldots, n$.

Task: Find assignments of the jobs to the machines s.t. the maximum completion time (completion of the latest job) is minimised.

This is an NP-hard problem. $T = \sum_{j=1}^{n} p_j$, obviously $v_opt(x) \geq \frac{T}{2}$ in case of equally loaded machines.

We call a job j large if $p_j \geq \varepsilon T$, otherwise small. (ε is our target precision, we aim for a $(1+\varepsilon)$-approximation).

Observation: There are at most $\lfloor \frac{1}{\varepsilon} \rfloor$ large jobs.

We look at all $2^{\lfloor \frac{1}{\varepsilon} \rfloor}$ possible assignments of the large jobs to the 2 machines. For each of these assignments we now distribute the small jobs by using the following very simple least loaded heuristic:

1. Go through the small jobs (in whatever order we want)
2. assign the jobs one by one to the machine which currently has the smaller load (ties are broken arbitrarily).
3. $\rightarrow$ we get $2^{\lfloor \frac{1}{\varepsilon} \rfloor}$ solutions and return the best among them.

Running time: $O(2^{\frac{1}{\varepsilon}} \cdot n)$, polynomial in n, not polynomial in $\frac{1}{\varepsilon}$, ok for PTAS.
Still to show: we und up with a $(1+\varepsilon)$-approximation, $v(x,s) \geq v_{opt}(x)$

Proof. Consider an optimal solution and consider the corresponding assignment of the large jobs (of the full problem) to M_1 and M_2. (This assignment is also considered by us since we consider all possible ones)
Cases

1. All small jobs end up on one of the machines (our approach cannot make anything wrong here)
2. Not all ...

   Here, the maximum difference of the two finishing time on M_1 and $M_2 \leq \varepsilon \cdot T$.

   $\rightarrow$ maximum difference of the finishing time of the machine which takes longer and $\frac{T}{2}$ (lower bound for optimal value) is bounded by $\varepsilon \frac{T}{2}$.

   $\Rightarrow v(x,s) \leq \frac{T}{2} + \frac{\varepsilon T}{2} = (1+\varepsilon)\frac{T}{2} \leq (1+\varepsilon) v_{opt}(x)$

□

Drawback: running time is exponential in $\frac{1}{\varepsilon}$, thus not practical

9.5 FPTAS Example: Knapsack Problem

Given: Set of n items. Item j has weight w_j and profit p_j, $j = 1, \ldots, n$ (Knapsack XXX

Task: Find subset $J \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in J} w_j \leq c$ and $\sum_{j \in J} P_j \to \max$

Classical dynamic programming approach for KP:

- Define $F(K, y) := \left\{ \max \sum_{j=1}^{k} p_j x_j \text{ such that } \sum_{j=1}^{k} w_j x_j \leq y, x_j \in \{0,1\}, j = 1, \ldots, n \right\}$
  for $k = 1, \ldots, n$, $y = 0, \ldots, b$.
  $F(n, b)$ delivers optimal value of KP
- Classical recursion $F(k+1, y) = \max\{F(K, y), F(K, y - w_{k+1}) + p_{k+1}\}$
- $F(n, b)$ can be computed in $O(nb)$ time.

Consider the following somehow "dual" approach:

- Define $M(k, v)$ as the minimum Knapsack capacity with which we can obtain a total profit of v by considering items $1, \ldots, k$.
  $k = 1, \ldots, n$, $v \in \{1, \ldots, \underbrace{v_{max}}_{\text{max. poss. profit}}\}$
  $v_{max} \leq \sum_{j=1}^{n} p_j$
- $M(k+1, v) = \min\{M(k, v), M(k, v - p_{k+1}) + w_{k+1}$
- Init $M(0, v) = \infty \; \forall v > 0$
  $M(i, v) = \infty \; \forall v < 0$
- An optimal solution of KP can be optained in $O(n \cdot V_{max}) = O(n^2 p_{max})$
  $p_{max} = max p_j$, $j = 1, \ldots, n$.

Idea: apply scaling (includes rounding) to the profits → new, easier Knapsack instance
Consider the new KP instance with n items with the old weights, but changed profits: replace p_j by $\tilde{p}_j = \lfloor \frac{p_j}{t} \rfloor$ where t will be an appropriately chosen scaling factor.
The Knapsack capacity stays the same.