Theoretical Computer Science 2

lecture notes, university of technology Graz

Lukas Prokop

January 20, 2014

Contents

Date. 2nd of Oct 2013

0.1 Structure of this course

Evaluations:

- 2-3 board presentations
- oral exam

Contents:

- Extended view on complexity theory and modern applications
- Focus: Algorithms, Optimization problems, cryptography

2 sections:

1. Complexity theory
    - randomized complexity classes
    - polynomial hierarchy
    - interactive protocols (zero knowledge systems)
    - probabilistically checkable proof (PCP theorem)
2. approximation of complexity (complexity theory of heuristic algorithms)

0.2 Literature

- C. Papadimitriou, "Computational complexity", 1994 [old, classic].
- I. Wegener, ,,Komplexitätstheorie (Grenzen in der Effizienz von Algorithmen)", 2003 [modern, non-comprehensive].
- S. Arora, B. Barak, Computational complexity (A modern approach), 2009 [modern, exhaustive].

1 Randomized complexity classes

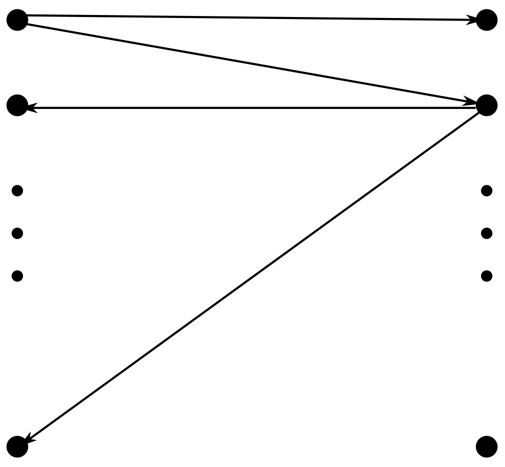
1.1 Problem: Evaluation of symbolic determinants for a bipartite graph

1.1.1 Problem and approach

$G = (U, V, E)$ is an undirected, bipartite graph with $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$.

Question: Does a perfect matching in the bipartite graph G exist?

Note. This question can be answered in polynomial time with a deterministic algorithm. We take another approach.



Consider a matrix $A(G) = (a_{i,j})$ for graph G. We distinguish 2 cases:

$$a_{i,j} = \begin{cases} \underbrace{x_{i,j}}_{\text{symbolic variable}} & \{u_i, v_j\} \in E \\ 0 & \{u_i, v_j\} \notin E \end{cases}$$

S_n is the set of permutations over $\{1, \dots, n\}$.

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

$$\sigma(\pi) = \begin{cases} 1 & \pi\text{is an even permutation} \\ -1 & \pi\text{is an odd permutation} \end{cases}$$

Observation:

1. Non-zero terms of the second factor (the product in $\det A$) correspond to a perfect matching.
2. G has a perfect matching if the $\det A(G) \neq 0$ (zero function).

Remarks:

- Consequence: An alternative method to solve the question above is based on the evaluation of $\det A(G)$.
- We gain interest in the evaluation of a symbolic determinant.

For actual values of $x_{i,j} \in \mathbb{Q}$ we can evaluate $\det A(G)$ in polynomial time (eg. via gaussian elimination). For the symbolic determinant evaluation no algorithm is known (actually, the question if a specific coefficient is the coefficient of the symbolic non-zero determinant, is an NP-complete problem).

In our case we want to know whether or not $\det A(G) = 0$.

1.1.2 Idea and Lemma

Select a random configuration for $x_{i,j}$ and evaluate the resulting determinant. If determinant is 0, then we have the zero function or a zero-point in the function. In the other case G has a perfect matching (result: YES). In the first case, we have to proceed with further evaluations.

Lemma 1.1: Assume $p(z_1, z_2, \dots, z_m)$ to be a polynomial with m variables (must not be the zero polynomial). u is the degree of the variables $\leq d$. Furthermore $M > 0$ and $M \in \mathbb{Z}$. It is fact that the number of m-tuples $(z_1, \dots, z_m) \in \{0, 1, \dots, M-1\}$ with $p(z_1, \dots, z_m) = 0$ is smaller or equal to M^{m-1} (so this is the upper bound for the null point number).

Proof of the Lemma: Complete induction to m.

$m = 1$: Number of null points $\leq d$ (with respect to the fundamental theorem of algebra) for a polynom of 1 variable of degree $\leq d$.

$m-1 \to m$: Polynomial p in $z_1, \dots, z_m$. Consider p to be a polynomial in z_m with coefficients which are in $z_1, \dots, z_{m-1}$.

Remarks:

- p results in evaluation of “gzz” test tuple 0.

Cases:

1. The coefficient with highest degree of z_m in p is 0.
   Apply induction to the coefficient of z_m with highest degree. The coefficient is a polynomal in $z_1, \dots, z_{m-1}$. Can therefore $\leq (m-1)$ of M^{m-2} permutations of $(z_1, \dots, z_{m-1})$. In combination with the selection of z_m (M possibilities). $\leq (m-1)$ of M^{m-1}.
2. Is not 0.
   We have a polynomial of degree $\leq d$ in z_m such that $\leq d$ null points can exist for each combination of $z_1, \dots, z_{m-1}$. Therefore $\leq d\ M^{m-1}$ further null points of p. Addition of $(m-1)$ in M^{m-1} with d with M^{m-1} results in $\leq m \cdot d$ with M^{m-1}. So this lemma is proven.

Consider the following algorithm: Choose m random integers $(i_1, \dots, i_m)$ for the assignment of $x_{i,j}$ with $\{i, j\} \in E$. (here $m = |E|$) with $x_{i,j} \in \{0, \dots, M-1\}$ whereby in the case of a matching $M = 2m = 2 \cdot |E|$.

Evaluate $\det A(G)$ for $x_{i,j}$ assignment $(i_1, \dots, i_m)$.

If $\det A(G)(i_1, \dots, i_m) \neq 0$ then answer “YES, G has a perfect matching”. If $\det A(G)(i_1, \dots, i_m) = 0$ then answer “I know know” (we need further repetitions).

Answer Yes with no error and “I don’t know in some cases”: Is an algorithm of complexity class RP.

1.1.3 Analysis of the algorithm

Properties of the algorithm:

- If the answer is Yes (there exists an perfect matching) the answer is always correct (no false positives).
- If the graph does not have a perfect matching, the answer of the algorithm is not complete. Consider the answer “No” if the number of repetitions with the answer “I don’t know” exceeds some threshold (therefore, false negatives are possible).

What's the probability that we get a false negative?

- In the case of matching we can see that the error probability (applying the algorithm once) is $\leq \frac{1}{2}$ (recognize that $d = 1$ here). k applications: $\leq \frac{1}{2^k}$.
- This is a Monte-Carlo algorithm.

1.2 Problem: SAT problem

2-SAT (polynomial evaluable with flow theory) and 3-SAT (NP-complete). 2 or 3 literals per clause.

Input: A boolean equation ϕ in conjunctive normal form.
Problem: Is ϕ satisfiable?

Consider the following algorithm for SAT:

1. Start with a random assignment T for the variables $x_1, \ldots, x_n$.
2. Repeat r times:
    (a) Is satisfiable? Answer yes.
    (b) c is one of the clauses which are not satisfied. Take one literal and negate its variable in the assignment.
3. Answer "no" or "probably no".

Is also a Monte-Carlo-Algorithm. Probability for false negatives? How to choose r?

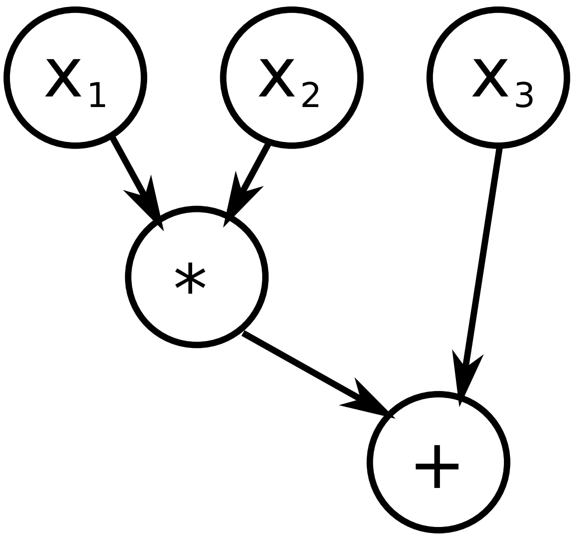
Is a polynomial upper bound of r (in regards of the number of clauses) enough for a practical error probability? (exponentially, it is easily feasible)

- No, even for 3-SAT there are known instances of the problem where a polynomial r fails.
- For 2-SAT for $r = 2n^2$ we have an error probability of $\leq \frac{1}{2}$. Therefore we have a Monte-Carlo algorithm for 2-SAT.

Date. 7th of Oct 2013

1.3 Problem: Identity test for polynomials

Polynomials are represented here as equations of an algebraic circuit (instead of the operators $\wedge, \vee$ and $\neg$ in classical booleans circuits, we have $+$, $-$ and $\cdot$) with n variables $x_1, x_2, \ldots, x_n$ represented as special directed graph.



$$x_1x_2 + x_3$$

All vertices (which are not sources and are not inner vertices) do have in-degree 2 und are annotated with operators $\{+, \cdot, -\}$.

1.3.1 Sinks

Remarks:

- Extensions are possible.
- Such a circuit creates a polynomials in $x_1, x_2, \ldots, x_n$.

Is ZEROP the complexity class of algebraic circuits, which evaluate a polynomial (which is identical) to a zero-polynomial.

$$c = c' \quad \Leftrightarrow \quad c - c' = 0$$

This is why the name ZEROP was chosen. A special case of this problem is exercise 1, we discussed previously.

Problem: Detection of ZEROP is non-trivial (ie. is a polynomial P $\in$ ZEROP).

Consider for example
$$\prod_{i=1}^{n}(1+x_i)$$
This is a very compact description, but the circuit has size $2n = \mathcal{O}(n)$ but polynomial has 2^n terms.

For this problem no deterministic, efficient algorithms is known, but a nice Monte-Carlo algorithm.

We can use the following Lemma 1.2 (analogous to Lemma 1.1):

> Is $p(x_1, \ldots, x_n)$ a polynomial of total degree $\max \tilde{d}$. For example $x_1^2 x_2^3 x_3 x_4^7$ has max-degree 13 for all terms.
>
> S is a finite set of integers. If $a_j, \ldots, a_n$ is selected randomly out of S.
>
> In this case:
> $$P(p(a_1, \ldots, a_n) \neq 0) \geq 1 - \frac{\tilde{d}}{|S|}$$

A circuit C of size m contains $\leq m$ multiplications and therefore degree $\leq 2^m$.

Idea for probabilistic algorithm:

1. Select random integers for $x_1, \ldots, x_n \in \{1, \ldots, 10, \ldots, 2^m\}$ (requires $\mathcal{O}(nm)$ random bits).
2. We evaluate C for those integers: y.
3. If $Y = 0$, answer "Yes". Else "I don't know".

If $C \in$ ZEROP, then we accept C always. If $C \notin$ ZEROP and we do not answer with "NO" for $y \neq 0$, we have a error probability. We reject $\leq \frac{1}{10} \cdot$ (Mil probability $\geq \frac{9}{10}$).

Problem: Degree can be at maximum 2^m. y and the intermediate results can grow up to $(10 \cdot 2^m)^{2^m}$. An exponential number of bits is necessary for representation. Therefore we would like to switch to modulo calculations.

We evaluate $C \mod k$ where k is randomly selected from $\{1, \ldots, 2^{2m}\}$ (further constraints are mentioned later). Instead of y, we get $y \mod k$. If $y = 0$, then also $y \mod k = 0$.

	failure/error probability (bounded)	failure/error probability (not bounded)
Two-sided error	BPP	PP
One-sided error	RP, co-RP RP = Monte-Carlo algorithms	NP, co-NP
No error, but failure is possible	$ZPP = RP \cap co - RP$ (Las Vegas Algorithms)	$NP \cap co - NP$
No error, no failure	P	

Table 1: Probabilistic complexity classes overview

If $y \neq 0$, then we can show that k (with probability $\geq \delta = \frac{1}{4m}$) does not split. This probability is small enough that $\mathcal{O}(\frac{1}{\delta})$ are enough repetitions to gain our result. We do only accept a^k, if for all runs the result was 0.

Is $y \neq 0$ and $B = \{p_1, \ldots, p_n\}$ the set of distinct prime factors of y. This satisfies that K is not a prime number of B is given with probability $\geq \delta$. With the prime number theorem, we can conclude that for sufficient large m the number of prime numbers in $\left\{1, \ldots, 2^{2m}\right\}$ is $\geq \frac{2^{2m}}{2 \cdot m}$.

y can have $\max \log y \leq 5, m \cdot 2^m = o(\frac{2^{2m}}{2m})$ prime factors.

For sufficiently large m it is given that the number of $k \in \left\{1, \ldots, 2^{2m}\right\}$ such that k is prime and $k \notin B$: $\geq \frac{2^{2m}}{4m}$ with probability $\delta = \frac{1}{4m}$ a random k has the desired properties.

1.4 Problem: Primality testing

In the meantime a deterministic polynomial primality test is known (Agrawal, Kayal, Saxena). For a few years only randomized primality tests (which represent Monte Carlo algorithms) were known.

Given: n is odd
Question: Is n not a prime number?

2 Probabilistic complexity classes overview

A quick overview: Definitions for this topic. See table 1.

Now we would like to formalize Monte-Carlo algorithms. Monte-Carlo turing machines are special cases of probabilistic turing machines. Probabilistic turing machines are a generalization.

3 Monte-Carlo turing machines and RP

Definition. N is a polynomial, non-deterministic turingmachine. We assume for N that all computations for an input x take the same number of steps. This number has to be polynomial in $|x|$, because N is a polynomial TM. Furthermore in every step there are exactly 2 non-deterministic decision possibilities.

L is a language. N is called Monte-Carlo turing machine for L iff N satisfies the definition above and the computations for an input of size N are finished in $p(n)$ runtime (with p as a polynomial) . Furthermore it satisfies:

- If $x \notin L$ then all computations of N for x end with the result "No" (no false positives).
- If $x \in L$ then more than half of the evaluations for x result in "Yes". (So the probability of false negatives $\leq \frac{1}{2}$.) ($\frac{1}{2}$ can be adjusted; is not fixed.)

RP (randomized polynomial) is a historical name and does not really distinguish it from other classes.

The set of all languages for which there exists a Monte-Carlo turing machine is called RP.

Remark:

- This concept corresponds to the informal concept of Monte-Carlo algorithms (of our exercises).
- If we replace the allowed error probability for false negatives to a value $\neq \frac{1}{2}$, but < 1, no new complexity class is created.

Repetitions:

$$1 - \epsilon \text{ with } \epsilon < \frac{1}{2} \text{ is the error probability of false positives}$$

For k repetitions: Error probability for false negatives is $(1-\epsilon)^k$. Choose k:

$$k = \left\lceil -\frac{1}{\log_2(1-\epsilon)} \right\rceil \text{ probability for false negatives is } \leq \frac{1}{2}$$

Runtime $k \cdot p(n)$ is polynomial for polynomial k.

RP lies between P and NP. co-RP is the complement of RP (ie. the result "Yes"/"No"/"I don't know" is exchanged).

4 Complexity class ZPP

ZPP (polynomial randomized with zero probability of error)

$$\text{ZPP} = \text{RP} \cap \text{co-RP}$$

In other words, there exists a Monte-Carlo algorithm which does not return any false positives and an analogon which does not return false negatives (no error, but failure is possible). This is the definition of Las-Vegas algorithms.

If probability of false positives and false negatives $\leq \frac{1}{2}$ (after k repetitions).

5 Complexity class PP

PP (probabilistic polynomial) is not defined algorithmically.

Input: We consider MAJSAT (majority SAT). Given is a SAT equation with n variables.
Problem: Is is true that the majority of $2^n \geq 2^{n-1} + 1$ assignments satisfy the equation?

It is unknown whether or not MAJSAT $\in$ NP (how should the polynomial certificate look like?). It is even more unlikely that MAJSAT $\in$ RP.

Definition. (PP) A language L is in PP, iff there is a non-deterministic polynomial TM N (N satisfies the basic constraints mentioned above) such that for all inputs x it states that

$$x \in L \quad \Leftrightarrow \quad \#(\text{Computations of N for x result in "YES"}) > \frac{1}{2}$$

Therefore this turing machine has an inherent majority evaluation capability.

Remark: Definition of PP is syntactically, but not semantically. We can easily show MAJSAT $\in$ PP and MAJSAT is PP-complete.

Date. 9. Okt 2013

5.1 NP $\subseteq$ PP

Proof. $L \in$ NP. N is a non-deterministic turing machine for L. We construct another turing machine N', which decides L with a majority vote. N' almost looks alike N. We introduce a new Start state and another branching (non-deterministic decision), which is executed in the Start state.

Start state - non-deterministic choice which branch to take:

1. Evaluation with N
2. Randomized selection of one evaluation path (for all inputs x of same size answer always with "YES").

Consider input x. x takes $p(|x|)$ steps in N (fundamental assumption as always).

$$2^{|x|} \text{ evaluation paths of N for x}$$

$$2^{|x|+1} \text{ evaluation paths of N' for x}$$

A majority of evaluations for N' for x ends with "Yes". This corresponds to: more than 1 evaluation of N for x ends with "YES". This corresponds to $x \in L$.

We can conclude $L \in$ PP.

Remarks:

- PP is closed under complement.
- PP is (in difference to our previous mentioned complexity classes) not algorithmically defined.

6 The class BPP

Definition. BPP (bounded error probabilistic polynomial) contains all languages L for which there is a non-deterministic polynomial turing machine N (without further constraints of the assumptions with our fundamental assumptions) such that:

- For all inputs x with $x \in L$ ends with probability $\geq \frac{3}{4}$ of the evaluations of N of x with acceptance.
- For all inputs x with $x \notin L$ ends with probability $\geq \frac{3}{4}$ of the evaluations of N of x with rejection.

Remarks:

- Any value $> \frac{1}{2}$ and < 1 can be used instead of $\frac{3}{4}$.
- For $\frac{3}{4}$: $\frac{1}{4}$ error probability for false positives and false negatives.
- RP $\subseteq$ BPP (Monte-Carlo algorithm with error probability $\frac{1}{2}$ repeated twice in different direction)
- co-RP $\subseteq$ BPP

- BPP $\subseteq$ PP (can easily be derived from the definition; has fewer constraints)
- Open issues:
  - BPP $\subseteq$ NP?
  - $\exists$ BPP-complete problems?
- BPP is closed under complement.
- BPP is basically a model for "*practical* polynomial randomized algorithm".

All these complexity classes require some concept of probability; for example random bits set on the turing machine. Question in practice:

Assume we don't have any fair dice/coins (probability(head) $\neq \frac{1}{2}$). Does this define a new complexity class?

You can create a fair dice with polynomial effort using an unfair dice. So RP and BPP (etc) are not influenced.

Take a coin with p(head) $= \rho$ with $\rho \neq \frac{1}{2}$. The following two lemmas apply:

1. (Lemma 1.3) A coin with p(head) $= \rho$ can be simulated by a probabilistic, polynomial turing machine in expected time $\mathcal{O}(1)$, if the i-th bit of ρ is determinable in polynomial time in i. Notice that the last assumption is only relevant for irrational numbers.
2. (Lemma 1.4) A coin with p(head) $= \frac{1}{2}$ (fair coin) can be simulated with a coin of p(head) $= \rho$ in expected time $\mathcal{O}\left(\frac{1}{\rho(1-\rho)}\right)$.

6.1 Proof of Lemma 1.3

Is $0.p_1p_2p_3\ldots$ a binary fractional representation of ρ. The probabilistic turing-machine creates a sequence of random bits $b_1b_2\ldots$ step by step, whereby bit b_i is created in step i. If $b_i < p_i$ then the turing machine returns "head" as result and stops. If $b_i > p_i$ then the turing machine returns "bottom" as result and stops. In any other case, go on (increment i).

If turing machine reaches bit $i+i$, then

$$b_j = p_j \quad \forall j \in \{1, \ldots, i\}$$

The probability for this is defined by $(\frac{1}{2})^l$.

The probability that the result is head is $\sum_i p_i \cdot \frac{1}{2^i} = \rho$. Expected runtime is $\sum_i i^c \cdot \frac{1}{2^i}$ for constant c.

6.2 Proof of Lemma 1.4

We construct a turing machine which simulates (using the possibility to use a coin with p(head) $= \rho$ multiple times) a fair coin.

M throws pair of coins (2 coins) with bias ρ. Repeat these paired throws until 2 different results are given. We return "head" for the result head-bottom and "bottom" for bottom-head.

Probability for head-bottom is $\rho \cdot (1-\rho)$. Probability for bottom-head is $(1-\rho) \cdot \rho = \rho \cdot (1-\rho)$. Probability that turing machine stops in every step is $2\rho \cdot (1-\rho)$. The conditional probability for stopping in step k is equal for head and bottom. So we can a fair coin with p(head) $= \frac{1}{2}$. This proof does work for any value of ρ.

6.3 Remark for lemma 1.2

The "probabilistic turing machine" must have two transition function δ_0 ad δ_1. We decide in every step whether we want to use (with probability $\frac{1}{2}$) δ_0 or (with probability $\frac{1}{2}$) δ_1. In the general model for a probabilistic turing machine which is considered for efficient algorithms we allow expected polynomial time.

From this lemma it follows that (if we allow expected polynomial time[1]) we don't lose any properties for a biased coin. We can also show this for polynomial runtime rather than expected runtime (by splitting the bitstring into blocks).

7 Relations of probabilistic complexity classes

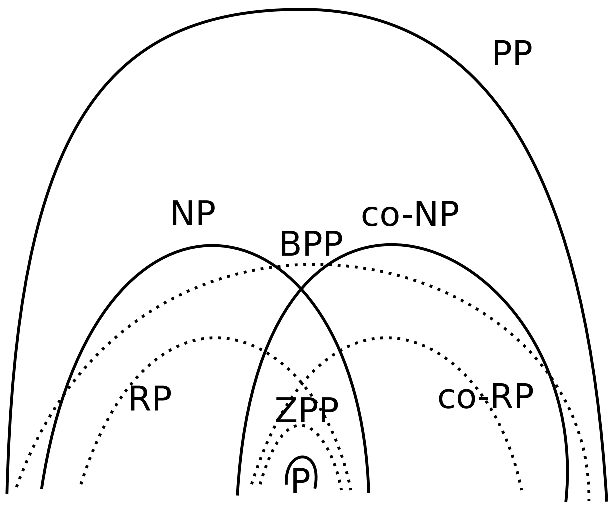
The position of BPP are partially unknown; partially we will discuss them later on.

The aspect of expected polynomial runtime is considered at a different place.

8 The class FP

Definition. (Complexity class of function problems for polynomial hierarchy). Many problems in practice are no decision problems (they result in complexer answers than "YES" or "NO"). Two examples are FSAT and TSP.

[1]which is realistic for practical applications



8.1 Problem: FSAT

Problem. Given is a SAT equation ϕ. If there is any satisfying assignment for ϕ we return such an assignment. Otherwise "No".

Assume that a polynomial SAT algorithm exists for the decision problem SAT, then we can solve FSAT in polynomial time: ϕ is a SAT equation with variables $x_1, x_2, \ldots, x_n$.

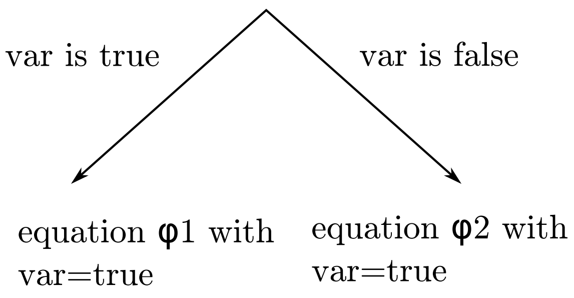
1. Apply SAT algorithm to ϕ. If answer is "no", then ϕ is not satisifiable and we return "there is no satisfying assignment".
2. (Successive fixation of variables.) We know that in one of the cases (true/false) the answer is "Yes" and an assignment can be found. Fix variable v in such a way (i.e. ϕ_1 or ϕ_2) that ϕ keeps satisfiable. Stop if all variables got fixated.

This approach takes $\mathcal{O}(n)$ SAT calls.

8.2 Problem: Travelling Salesman Problem (TSP)

TSP with distance matrix of integers.

$$D = (d_{ij}) \quad d_{ij} \in \mathbb{Z}_0^+$$



The classical definition of TSP: Given is a distance matrix D and we want to find a hamiltonian path with minimum length/weight. The corresponding decision problem (TSP_{DEC}): $n \times n$ matrix $D = (d_{ij})$ with $d_{ij} \in \mathcal{Z}_0^+$ with upper bound d^* of costs. Is there any hamiltonian path with length $\leq d^*$?

Date. 14th of Oct 2013

Given: Distance matrix D.
Find: Hamiltonian tour.

Decision problem TSP_{dec}. We show that when a polynomial algorithm for TSP_{DEC} exists, then there is a polynomial algorithm for TSP:

Step 1 Determine the requested target function value (length of optimal tour).

Value of an optimal tour is in $\{0, 1, \ldots, 2^q\}$ with q as the coding length of the instance.

Binary search for the optimal value for the optimal value. Always ask: is there any tour with length ($\leq 2^{q-1}$)? We get a binary tree always with two branches "Yes" and "No". On the second level we ask is there any tour with length ($\leq 2^{q-2}$).

After a polynomial number ($\mathcal{O}(\log_2(20) = \mathcal{O}(p)$) of steps c^*, the length of the requested tour is known.

Step 2 Determine a required tour (a tour with length c^*). Consider the values of D in arbitrary order one after another. d_{ij} is the current item we look at. Then set d_{ij} temporarily to $c^* + 1$ (Idea: restrict usage of (i, j)) and ask whether there is a tour of length $\leq c^*$ for the current matrix.

If answer is no, then we have to use the current edge (i, j) and we set d_{ij} back to its old value. In the other case we keep d_{ij} as $c^* + 1$.

In the end (after $\mathcal{O}(n^2)$ steps) we know the optimal tour.

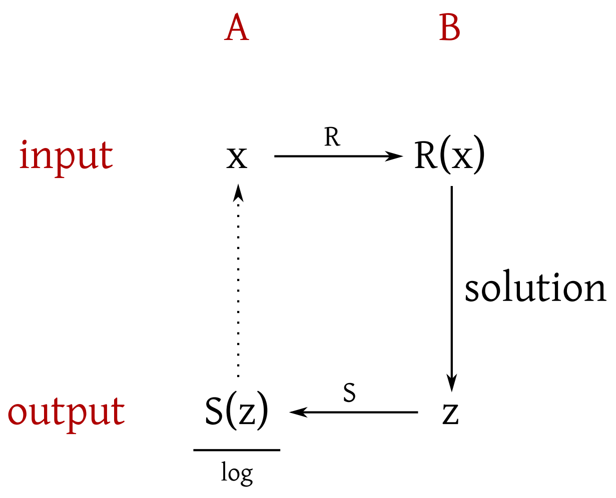
Formally L is a language in NP. We want to define the associated function problem to L: FL. For $L \in$ NP there is a binary relation R_L, which is determinable in polynomial time such that $\forall x \text{strings} \exists \text{string} y$ with $\underbrace{R_L(x, y)}_{(x,y) \in R_L} \Leftrightarrow x \in L$.

$R_L(x, y)$ is the certificate of an accepted instance.

Definition. (FL) Is $L \in NP$ the associated problem FL for L. Given a string x we have to find a string y such that $R_L(x, y)$ or answer that there is no such y.

Next task: Generalize the class / reduction term of decision problems for function problems.

Definition: A and B are two function problems. A is *reducible to* B, iff $\exists$ string functions R and S (both compute in polynomial space) such that:



x is an input string of an instance of A such that $R(x)$ is an input string of an instance of B. Is z a correct output for the instance of B with input $R(x)$ then $S(z)$ is a correct output for the instance with input x.

Remark: Any logarithmic boundary for space implies polynomial time.

Definition. A function problem A is says to be *complete for a class FC* of function problems iff $A \in$ FC and all problems in FC can be reduced to A.

We can show that FSAT $\in$ FNP is FNP-complete.

We want to consider a special class of function problems. They are problems for which there are always valid solution and as such the answer is never "No". Example for such a problem: $\exists$ always y with $R_L(x, y)$?

8.3 Factorization of integers

Given: A number $n \in \mathbb{N}$
Find: prime number decomposition

The corresponding decision problem is not interesting (Answer is always yes). Function problems are not trivial. It's an open question whether factorization is possible in polynomial time. Factorization is fundamentally different than FSAT where a decision problem cannot be solved efficiently.

Definition. (total functions) For such function problems there is always a valid solution.

Definition. We consider a problem L in FNP total, when for all strings $x \exists$ string $y \exists$ with $R_L(x, y)$.

Definition. (TFNP) partial class of all function problems with total functions.

8.4 Exercise 2

Given: n numbers $a_1, \ldots, a_n \in \mathbb{Z}^+$ with $\sum_{i=1}^n a_i < 2^n - 1$.
Find: $I, J \subset \{1, \ldots, n\}$ with $I \neq J$ such that $I \cap J = \varnothing$ (not always required) and $\sum_{i \in I} a_i = \sum_{i \in J} a_i$ (or answer No if there are no such sets).

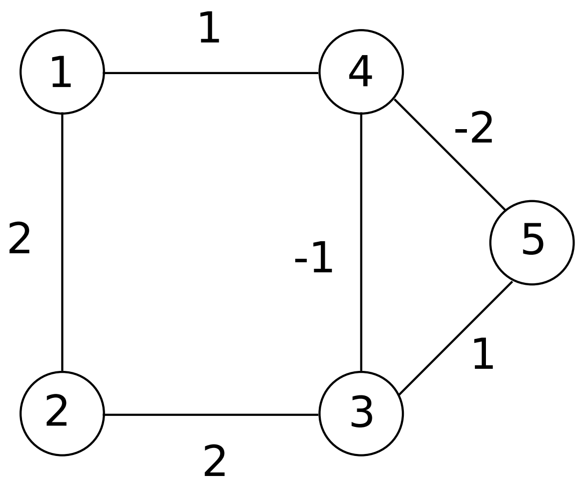
For the special case $\sum_{i=1}^n a_i < 2^n - 1$ there is always a solution due to the pidgeonhole principle. Without this constraint, the decision problem is complete.

8.5 Exercise 3

Given: Undirected graph G, edge weights $w_e \in \mathbb{Z}$ for all $e \in E$.

State function S: $V \to \{+1, -1\}$.

A vertex $i \in V$ is called "happy for S" iff $S(i) \cdot \sum_{\{i,j\}\in E} S(j) \cdot w_{ij} \geq 0$.



Find a state s in which all vertices are happy. Problem is known as happy net problem.

Claim. There always is such a state.
Proof. Potential function $\rho[s] := \sum_{\{i,j\}\in E} S(i) \cdot S(j) \cdot w_{ij}$.

Remark. i is unhappy for S.

$$\underbrace{S(i) \cdot \sum_{\{i,j\}\in E} S(j) \underbrace{w_{ij}}_{\in \mathbb{Z}} < 0}_{=-\delta \text{ with } \delta>0, \delta\in\mathbb{Z} \text{ and therefore } \delta\geq 1}$$

Transition $S \to S'$ with $S'(j) = S(j) \forall j \neq i$ and $S'(i) = -S(i)$ (flip) otherwise. We can show that:

$$\rho[S'] = \rho[S] + 2\delta$$

We come up with an algorithm from this:

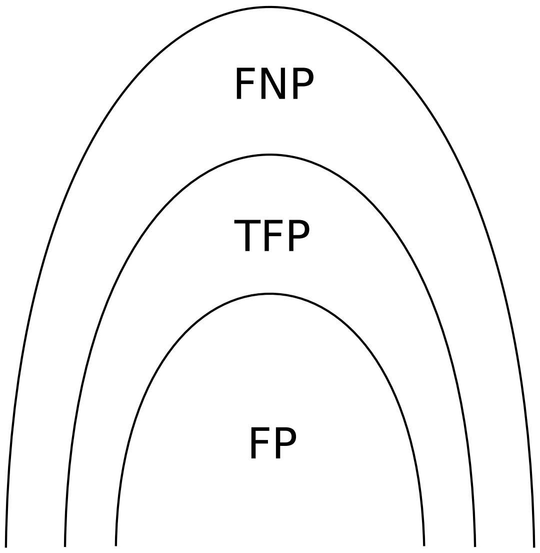
> Start with a random start state. As long as there are unhappy vertices, flip them. Because there are ρ values in $[-W, W]$ with $W = \sum_{\{i,j\}\in E} |w_{ij}|$ and ρ increases with $\geq 2^{\{i,j\}\in E}$, we have shown the finiteness of the algorithm.

This is also a total function problem. The algorithm has pseudo-polynomial runtime (because it depends on pseudo-polynomial property of W). It is an open question whether there exists a polynomial algorithm.

Given: Given an undirected graph ($G = (V, E)$) and a hamiltonian cycle Q in G. Find: A second hamiltonian cycle (but does not have to exist) (different from Q)..

For cubic graphs (3-regular[2]) there is always a second hamiltonian cycle (if there is at least one).

Is a total function problem and it is open whether there is a polynomial algorithm.



8.6 Variant 1 of TSP: $\mathbf{TSP_{DEC}}$

Given: $n \times n$ matrix D and a boundary c^*.
Question: Is there a tour with length $\leq c^*$?

8.7 Variant 2 of TSP: EXACT-TSP

Given: $n \times n$ matrix D, value c^*
Question: Does the optimal tour has length c^*?

[2] $d(v) = 3 \forall v \in V$

8.8 Variant 3 of TSP: TSP-COST

Given: $n \times n$ matrix D.
Find: Length of the optimal tour

8.9 Variant 4 of TSP: TSP

Given: $n \times n$ matrix D.
Find: optimal tour.

9 Complexity class DP

Definition. (difference polynomial) $L \in$ DP $\Leftrightarrow \exists$ languages $L_1 \in$ NP and $L_2 \in$ co-NP with $L = L_1 \cap L_2$.

Remark. DP $\neq$ P $\cap$ co-NP.

9.1 Examples for problems in DP

Examples:

- EXACT-TSP $\in$ DP.
  Split problem in $\leq c^*$ (in NP) and $\geq c^*$ (in co-NP) questions.
- SAT-UNSAT
  Given a SAT formular ϕ and ϕ'.
  Question: Is ϕ satisfiable or ϕ' not satisfiable?

Statement 2.1 SAT-UNSAT $\in$ DP-complete.

Proof:

1. SAT-UNSAT $\in$ DP.
   $L_1 = \{(\phi, \phi') : \phi$ is satisfiable$\}$.
   $L_1 \in P$ due to SAT $\in$ NP.
   $L_2 = \{(\phi, \phi') : \phi$ is not satisfiable$\}$.
   $L_2 \in$ co-NP due to co-SAT $\in$ co-NP ($L = L_1 \cap L_2$).
2. SAT-UNSAT is DP-complete.
   Is $L \in DP$ and L is reducible to SAT-UNSAT.
   $L \in DP \to \exists l_1 \in$ NP$, l_2 \in$ co-NP
   $\exists$ reduction R_1 of L_1 to SAT. $\exists$ reduction R_2 of $\overline{L_2}$ (complement) to SAT.

Construct reduction R with $R(x) = (R_1(x), R_2(x))$ of L to SAT-UNSAT. $R(x)$ is YES-instance of SAT-UNSAT $\Leftrightarrow R_1(x)$ is satisfiable, $R_2(x)$ is not satisfiable $\Leftrightarrow x \in L_1, x \in L_2 \Leftrightarrow x \in L$.

Statement 2.2 EXACT-TSP $\in$ DP-complete.

(repetition)

$L \in$ DP : $L = L_1 \cap L_2$ with $L_1 \in$ NP, $L_2 \in$ co-NP.

SAT-UNSAT $\in$ DP-complete.

Satz 2.2. EXACT-TSP is DP-complete.

Proof:

1. EXACT-TSP $\in$ DP has been proven above.
2. DP-Completeness (reduction of SAT-UNSAT)
   We start from the classical reduction of 3SAT to the hamiltonian cycle problem. Using the equations ϕ, ϕ' of the SAT-UNSAT instance we get 2 graph G and G' such that G and G' posess a hamiltonian path $\Leftrightarrow \phi$ and ϕ' is satisfiable

Remark: Idea of reduction 3SAT to hamiltonian path problem: Given: 3SAT equation $\phi = (x_1, x_2, \ldots, x_n)$ and clauses $c_1, c_2, \ldots, c_n$ with ≤ 3 literals per clause.

Construction of a graph $G = R(\phi)$ which has a hamiltonian path $\Rightarrow \phi$ is satisfiable.

Construction consists of the components, which are connected at the ends.

The image shows the selection gadget for each variable.

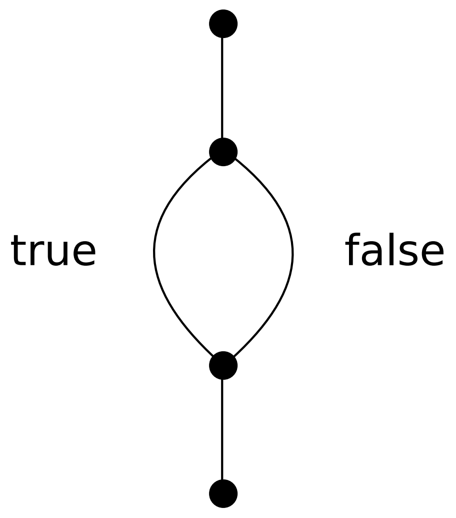
- Inner nodes only occur it the corresponding component.
- They relate to the outside.

There are always 2 possibilities to follow a path. The horizontal lines are followed by 2 paths where the paths can be described by a XOR (upper horizontal line is assigned to a different path than the corresponding lower horizontal line).

This ensures that the variable are kept consistent.

Clause gadget. without constraining the assumptions, 3 literals per clause.

The selection gadgets for x_1 to x_n are connected in series.



$(x_1 \vee x_2 \vee x_3)$	for hamiltonian path 1
$(x_1 \vee x_2 \vee x_3)$	for hamiltonian path 2

All $3m$ triangular nodes (of the m clause gadgets) and the last node of the "Kelle" of the variable gadgets and a new node 3 are connected by all possible nodes (complete subgraph). We finally add node 2 and edge $\{2, 3\}$.

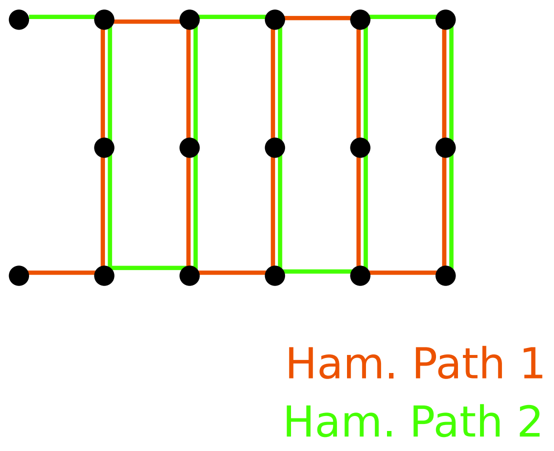
$G(\phi)$ is now finalized. We can show that the hamiltonian path $\exists \Leftrightarrow \phi$ is satisfiable. Such a hamiltonian path has to exist between 1 and 2 and without constrainting of our previous assumption, we start in 1.

(Further details shall be studies as home exercise)

9.2 Back to our EXACT-TSP construction

Independent of the satisfiability of ϕ and ϕ', G and G' contain a so-called broken hamiltonian path. This corresponds to an "almost satisfiable truth assignment" (all but one variable are satisfied). Introduce a new variable z and insert $\vee z$ to all clauses and another clause $\overline{z}$ (can be reduced to a 3SAT normal form). The new SAT equation requies an almost satisfying assignment (set $z = \top$).

If we apply the reduction of the variable-introduction, so we get an almost satisfying truth assignment T and its corresponding graph, which has an broken



hamiltonian path: We start in 1, traverse all variable gadgets according to T and traverse all clause gadgets. We interrupt this hamiltonian path only for the second-last clause.

So the path is interrupted at exactly for one edge.

Apply this construction to ϕ and ϕ'; G and G' are created where each of them contains a broken hamiltonian path.

In the left graph we define the edge weights to get an EXACT-TSP instance.

Set

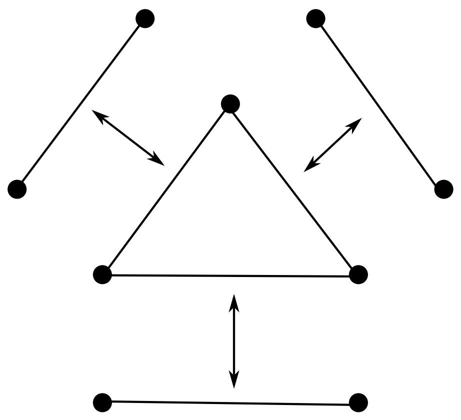
$$d_{ij} := \begin{cases} 1 & \text{if } \{i,j\} \text{ is an edge in G or edge in G'} \\ 2 & \text{if } \{i,j\} \text{ is no edge in G but i,j are vertices of G} \\ 3 & \text{else} \end{cases}$$

Question: What is the length of an optimal tour?

Case 1: ϕ and ϕ' are satisfiable. G and G' have a hamiltonian path (together: hamiltonian cycle). So we only have 1-edges (optimal tour has length n).

Case 2: ϕ and ϕ' are not satisfiable. We get broken satisfiable paths which have to be fixed. This has additional cost of $+1+2$. Optimal tour has length $n+3$.

Case 3: ϕ is satisfiable, ϕ is not satifiable



Optimal tour has length $n + 2$

Case 4: ϕ is not satisfiable, ϕ is satisfiable
Optimal tour has length $n - 1$

(ϕ, ϕ') is Yes-instance of SAT-UNSAT $\Leftrightarrow$ in our EXACT-TSP instance ($c^* = n + 2$) exists an optimal tour with length $n + 2$.

9.3 Other problems

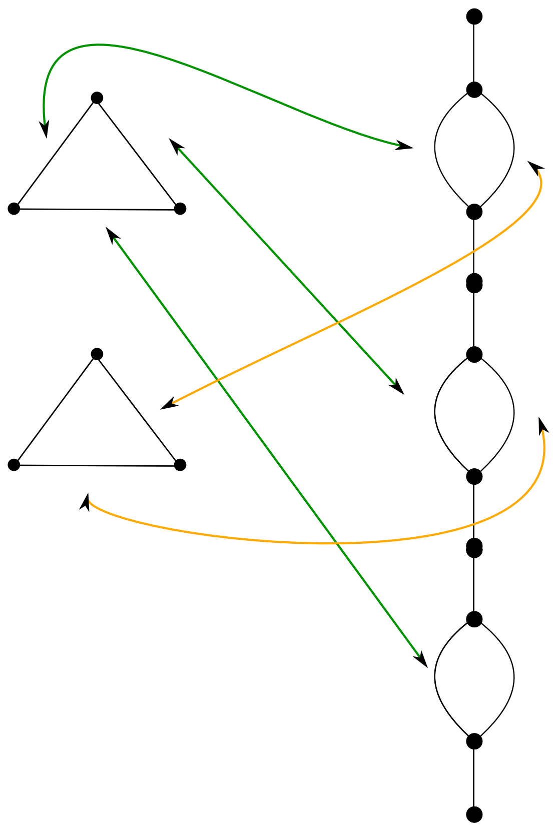
Many other NP-hard combinatorial optimization problems lead in the EXACT-version to DP-complete problems. DP is the natural class of EXACT-COST.

Other problems of DP:

Critical SAT Given a SAT equation ϕ.
Is it true that ϕ is not satisfiable, but by deleting one clause we get a satisfying one?

Critical Hamiltonian Path Given an undirected graph $G = (V, E)$
Is it true that G does not have a hamiltonian path, but adding one arbitrary edge leads to the existence of such.

Unique SAT Given a SAT equation ϕ
Does ϕ has exactly one satisfying assignment?



All problems above (except Unique-SAT) are DP-complete. For Unique-SAT it is unknown.

Theorem 2.3. (A first step towards polynomial hierarchy.) A different approach to DP: Use a SAT oracle and one co-SAT oracle. Answer “Yes” whenever first answer (L_1) is yes and second answer (L_2) is “No”.

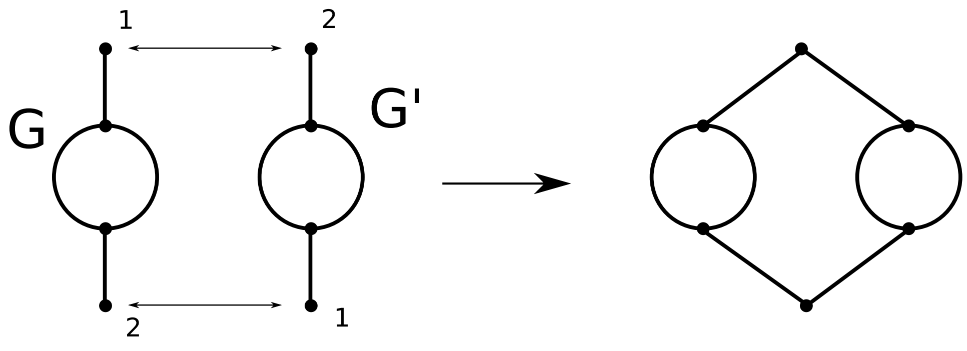
Definition. A turing machine $M^?$ with an oracle is a turing machine with a specfici string (query string of oracle call) and 3 specified, additional states: $q_?$ is the state of the oracle call.

q_{Yes}	State for Yes-answer of the oracle
q_{No}	State for No-answer of the oracle

$A \subseteq \sigma^*$ is a language.

$M^?$ with A as oracle runs like a traditional turing machine except for the transition of the query state.

$$M^A(x) \ldots \text{applies oracle TM } M^? \text{ with oracle A to input x}$$



Instead of the single language, we also consider language classes. P^{SAT} is a generalization of DP and has an polynomial number of oracle call available. P^{SAT} corresponds to P^{NP}, which can be achieved by oracle turing machine with oracle of NP. Polynomial number of steps (including number of oracle calls, excluding time for running the oracle). Analogous: FP^{NP}.

Question: Are there any FP^{NP}-complete problems?

9.4 MAX-OUTPUT

Given a non-deterministic turing machine N with input string 1^n. N behaves in such a way that for this input and for an arbitrary sequence of the non-deterministic $\mathcal{O}(n)$ steps stops. It returns a binary string of length n as output.

Given: Greatest output of N (interpreted as binary number)

Theorem 2.3 MAX-OUTPUT is FP^{NP} complete.

Will be a helping subproblem to show FP^{np} completeness of natural problems.

Proof: First we show that MAX OUTPUT $\in \text{FP}^{\text{NP}}$. Idea: Binary search. Iterate question: Is there an output greater $\geq x$? (or $> x$) (for integers $x \in \{0, \dots, 2^{n-1}\}$).

We determine the optimal value in $\mathcal{O}(\log n)$ (polynomial number of) steps.

Each question is a question in NP. Therefore: $\mathrm{FP}^{\mathrm{NP}}$.

Secondly, we get a little bit more technical: We show $\mathrm{FP}^{\mathrm{NP}}$ completeness. F is a function of strings to string with $F \in \mathrm{FP}^{\mathrm{NP}}$. There exists an oracle turingmachine $M^?$ which evaluates F with a polynomial time boundary. Thus, $M^{\mathrm{NP}}(x) = F(x)$ or correspondingly $M^{\mathrm{SAT}}(x) = F(x)$.

We need a reduction from F to MAX-OUTPUT: We need a function R converting F to MAX-OUTPUT ($R(x)$) and a function S to convert the output of R to $F(x)$ (solution, output).

We need two functions R and S such that

- R and S are determinable with logarithmic space requirements..
- For all strings x, $R(x)$ is an instance of MAX-OUTPUT.
- S applied to the output of MAX-OUTPUT for an instance $R(x)$ which evaluates to $F(x)$.

Regarding the construction of R We have to build a non-deterministic TM N and input 1^n. We set $n = p^2(|x|)$ with $p(.)$ as the polynomial time boundary of TM M^{SAT} (informally this definition requires that sufficient time is available to simulate M^{SAT}).

Regarding N: B starts with an input 1^n and creates x as string and simulates M^{SAT} to x afterwards.

This simulation M^{SAT} is deterministic and correct for all operations, that are no calls to the oracle. Assume that M^{SAT} stops at the first oracle call. An oracle call corresponds to the question whether this SAT equation ϕ is satisfiable.

N guesses the answer to this question and sets $z_1 = 1$, if the guesses answer is "Yes" and $z_1 = 0$ otherwise. In the case of $z_1 = 0$, N continues the simulation with state q_{No}. If $z_1 = 1$, then N guesses an assignment T_1 for the variables in ϕ_1 and checks whether T_1 satisifies ϕ_1. If this test succeedes, the simulation is continued in state q_{Yes}. Otherwise to stop and resign. N returns the string 0^n as output and halts (non-successful evaluation).

Application to all oracle calls:

$$\phi \ldots \text{equation in oracle call } i$$

$$z_i \in \{0,1\} \text{ analoguous to } z_1$$

In the case that the turingmachine does not halt yet ($\{z_1, z_2, \ldots, z_r\}$ as the set of oracle calls), N writes string $z_1, z_2, \ldots, z_r$ and possibly padded with zeros at the end for length n and output of $M^{\mathrm{SAT}}(x) = F(x)$.

- A partial set of the successful evaluations are faulty simulations of M^{SAT}. ϕ_i can be satisfiable, but N probably uses $z_i = 0$. In case $z_i = 1$ we don't have any faulty case. At this point stop for this problem.

Main Observation: Each successful evaluation which returns the greatest output corresponds to a correct simulation.

Proof of the Main Observation: Assume in a successful evaluation which returns the maximal output, $z_j = 0$ but ϕ_j is satisfiable.

Furthermore assume j is chosen minimally (i.e. no faulty evaluations beforehand). Therefore there must be another successful simulation of N, for which the current and all until the j-th oracle call do correspond, but now $z_j = 1$ and a satisfying assignment T_j for each ϕ_j is guessed. Afterwards all remaining oracle calls create correct simulations.

Output of the first successful evaluations: $z_1, \ldots, z_{j-1}, 0, \ldots$.
Output of the current evaluations: $z_1, \ldots, z_{j-1}, 1, \ldots$.

Thus the Main Observation is valid.

N therefore solves the MAX-OUTPUT problem and N can be constructed with a logarithmic space boundary. S is trivial, because $F(x)$ is written by N to the tape.

9.5 MAX-WEIGHT

Given: a SAT equation ϕ and a weight for each clause.
Find: Truth assignment in such a way that the sum of weights of satisfying clauses is maximal.

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

Proof:

1. Using binary search and a SAT oracle we can evaluate the greatest total weight of satisfiable clauses. With variable fixitation like in FSAT we get an assignment which has the target function value. Thus MAX-WEIGHT $\in \text{FP}^{\text{NP}}$.
2. Reduce MAX-OUTPUT to MAX-WEIGHT-SAT. The proof of the theorem by Cook uses a non-deterministic N and an input 1^n. A boolean equation $\phi(N, n)$ is constructed such that each satisfying assignment of $\phi(N, n)$ corresponds to a correct evaluation of N to 1^n.

We use this construction. All clauses of $\phi(N, n)$ are assigned a high weight value; for example 2^n.

We introduce new clauses. The weight are chosen in such a way that in every optimal solution of MAX-WEIGHT-SAT all clauses of $\phi(N, n)$ are satisfied.

$\phi(N, n)$ contains a variable for the symbol at each position for each string of N for every step. There are n variables $(y_1, y_2, \ldots, y_n)$. Those represent the positions in the output string if N halts. We add n clauses to $\phi(N, n)$. Each of them contains only one literal:

$$(y_i) \quad i = 1, \ldots, n$$

The weights of this clause 2^{n-i}. Remark: Those clauses together are weighted less than each $\phi(N, n)$.

Hypothesis: Every optimal solution of MAX-WEIGHT-SAT corresponds to a correct evaluation of N for 1^n such results in the greatest-possible output of N to 1^n.

This defines the R-function reduction of the proof. Coming to the S-function...

From an optimal solution of MAX-WEIGHT-SAT we can construct an optimal output for MAX-OUTPUT easily (even from the optimal target function value).

Also we can show that

$$\text{TSP} \in \text{FP}^{\text{NP}} \text{ complete}$$

Reduction (for example) via MAX-WEIGHT-SAT. A lot of other combinatorial optimization problems are FP^{NP}-complete. For example the Knapsack-problem is a weighted version of MAX-CUT and BISECTION-WIDTH. The following algorithms are not in FP^{NP} (they distinguish from the previous ones because those algorithms have a polynomial upper bound for the maximized target function value (for example number of edges of a clique $\leq n$):

- MAX-CLIQUE (find complete subgraph of undirected graph G with maximum cardinality).
- Unweighted MAX-CLIQUE (maximize number of edges)
- Unweighted MAX-WEIGHT-SAT (maximize number of satisfiable clauses)

For such problems (specifically MAX-CLIQUE) $\mathcal{O}(\log n)$ oracle calls are sufficient. We get new complexity classes:

$$\text{P}^{\text{NP}[\log n]} \qquad \text{FP}^{\text{NP}[\log n]}$$

A class of languages which can be decided by a polynomial oracle M to the input x with $\mathcal{O}(\log |X|)$ SAT-oracle calls.

Theorem 2.5 MAX-CLIQUE is $\text{FP}^{\text{NP}[\log n]}$-complete.

Other possible constraints in these classes: The polynomial number of oracle calls do not occur adaptively (therefore without knowledge of the previous oracle call results) The calls will get parallelizable (para-classes) [correctly denoted with a vertical =-sign at the bottom].

Theorem 2.6 $P^{\mathrm{NP}}_{\mathrm{PARA}} = P^{\mathrm{NP}[\log n]}$. $\mathrm{FP}^{\mathrm{NP}}_{\mathrm{PARA}} = \mathrm{FP}^{\mathrm{NP}[\log n]}$.

Proof of $P^{\mathbf{NP}}_{\mathbf{PARA}} = P^{\mathbf{NP}[\log n]}$

$$P^{\mathrm{NP}[\log n]} \subseteq P^{\mathrm{NP}}_{\mathrm{PARA}}$$

Consider an oracle-TM which makes $\mathcal{O}(\log n)$ (adaptive) NP-oracle calls. For the first call: 2 possible outcomes (depending on yes/no answer). For second call the same, etc. This gives us a complete decision tree describing $2^{k \cdot logn} \in \mathcal{O}(n^k)$ oracle calls and answers.

For the simulation with a non-adaptive Oracle-TM determine all $\mathcal{O}(n^k)$ oracle calls and answers beforehand. The correct decision path can be derived.

$$P^{\mathrm{NP}}_{\mathrm{PARA}} \subseteq P^{\mathrm{NP}[\log n]}$$

L is a language $\in P^{\mathrm{NP}}_{\mathrm{PARA}}$. L can be evaluated with a polynomial number of non-adaptive oracle calls. Firstly we determine with $\mathcal{O}(\log n)$ NP-oracle calls (binary search) the number $\overline{K}$ of Yes-answers for the given non-adaptive oracle calls.

Remark: The question whether the given set of SAT equations $\geq K$ is satisfiable is a NP-question.

Finally, are there $\overline{K}$ satisfying truth assignments for $\overline{K}$ of the n expressions such that if all others are not satisfiable (for us this is the case because of $\overline{K}$ max.) the oracle-TM halts with acceptance? $L \in P^{\mathrm{NP}[\log n]}$.

10 The polynomial hierarchy

$$\begin{gathered}
\delta_0 P = \Sigma_0 P = \Pi_0 P := P \\
\delta_{i+1} P := P^{\Sigma_i P} \\
\Sigma_{i+1} P := \mathrm{NP}^{\Sigma_i P} \\
\Pi_{i+1} P := \text{co-NP}^{\Sigma_i P} \\
\mathrm{PH} := \bigcup_{i \geq 0} \Sigma_i P
\end{gathered}$$

Remark: This definition is based on the oracle turingmachine concept. We will consider alternative definitions later on. Instead of $\Sigma_i P$ also Σ_i^P is denoted in literature.

10.1 Layer 1

$$\delta_1 P = P^P = P$$
$$\Sigma_1 P = \text{NP}^P = \text{NP}$$
$$\Pi_1 P = \text{co-NP}^P = \text{co-NP}$$

10.2 Layer 2

$$\Delta_2 P = P^{\text{NP}}$$
$$\Sigma_2 P = \text{NP}^{\text{NP}}$$
$$\Pi_2 P = \text{co-NP}^{\text{NP}}$$

Goal: Characterization of Σ_i and Π_i (recursive and non-recursive).

Theorem 2.7. L is a language and $i \geq 1$. We can state:

$$L \in \Sigma_i P \Leftrightarrow \exists \text{polynomial balanced relation } R$$

R is a polynomial balanced relation such that the language $\{x; y : (x, y) \in R\}$ originates from $\Pi_{i-1}P$ and $L = \{x : \exists y \text{ with } (x, y) \in R\}$.

Term "polynomial balanced". A binary relation R is called polynomial balanced iff $\exists k \geq 1$ such that $(x, y) \in R \Rightarrow |y| \leq |x|^k$.

Proof by induction (by i):

1. With $i = 1$, $\Sigma_1 P = \text{NP}$ The stated theorem results from the known characteristics of NP with a certificate. We know:

   $$L \in \text{NP} \Leftrightarrow \exists \text{in polynomial time decidabe, polynomial balanced relation R}$$

   $$\text{with } L = \{x : (x, y) \in R \text{ for some y}\}$$

   Induction introduction accomplished.

2. $i \geq 2$ and step $i - 1 \rightarrow i$

   - "$\Leftarrow$": There exists a relation R according to the theorem (right part of the equation) additionally $L \in \Sigma_i O = \text{NP}^{\Sigma_{i-1}P}$.
     Thus, we have to define a non-deterministic oracle turingmachine which uses an oracle from $\Sigma_{i-1}P$ and decides L.
     For the input x this TM guesses the associated y and asks $\Sigma_{i-1}P$ oracle, whether $(x, y) \in R$ (or actually— because R is a $\Sigma_{i-1}P$ relation—whether $(x, y) \notin R$).

- "$\Rightarrow$": $L \in \Sigma_i P$ (additionally with a corresponding relation R).
  Because $L \in \Sigma_i P$ there exists a non-deterministic oracle turingmachine $M^?$ with oracle $K \in \Sigma_{i-1}P$ such that $M^?$ decides L. Because $K \in \Sigma_{i-1}P$ there exists a polynomial balanced relation S with S decidable in Π_{i-2} such that $z \in K \Leftrightarrow \exists w : (z, w) \in S$.

3. We construct R with the help of S. We know $x \in L \Leftrightarrow$ there exists a valid, accepted computation of the oracle turingmachine M^k for x (several steps of M^k are oracle calls [return Yes/No answers]). The associated certificate is y.

4. For each oracle call z_i with answer "Yes" a certificate w_i with $(z_i, w_i) \in S$ is returned.

5. Define R has followed: $(x, y) \in R \Leftrightarrow y$ describes an accepted evaluation of M^k to x together with the certificates w_i for all oracle calls z_i with answer "Yes".

6. Additionally R has the desired properties.
   Hypothesis. $(x, y) \in R$ can be decided in $\Pi_{i-1}P$.
   Proof of hypothesis. First of all we have to test whether all steps $M^?$ (with k computed by the oracle) are valid. This is possible in deterministic polynomial time. Then we have to test for a polynomial number of pairs (z_i, w_i) whether $(z_i, w_i) \in S$. This can be computed in $\Pi_{i-2}P$ and therefore also in $\Pi_{i-1}P$. Because $k \in \Sigma_{i-1}P$, this question is a $\Pi_{i-1}P$ question.

7. So $(x, y) \in R \Leftrightarrow$ A sequence of questions of $\Pi_{i-1}P$ have the answer "Yes". Show in $\Pi_{i-1}P$.

Corollary 2.8. (recursive characteristic of $\Pi_i P$) L is a language and $i \geq 1$. Then: $L \in \Pi_i P \Leftrightarrow$ there exists a polynomial balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in $\Sigma_{i-1}P$ and $L = \{x : \forall y \text{ with } |y| \leq |x|^k : (x, y) \in R\}$.

Proof of Corollary 2.8. Is a direct result from Theorem 2.7.

We are looking into non-recursive characteristics now:

Corollary 2.9. (non-recursive characteristic for $\Sigma_i P$) L is a language and $i \geq 1$. Then $L \in \Sigma_i P \Leftrightarrow$ there exists a poly., balanced, in polynomial time decidable (i+1)-ary relation R with $L = \{x : \exists y_1 \forall y_2 \exists y_3 \dots Q y_i : (x, y_1, y_2, \dots, y_i) \in R\}$ with Q as quantifier ($\forall$ in even case, $\exists$ in odd case). This $\dots$ describes an alternating quantifier sequence.

Proof results from the repeated application of Theorem 2.7 and Corollary 2.8 and adhesion of certificates.

Remark:

- Analogously the non-recursive characteristic of $\Pi_i P$. But this sequence starts with an all-quantifier.

- An alternating introduction of the clases $\Sigma_i P$ and $\Pi_i P$ would be by an alternating quantifier usage in the relational representation.
- Another alternative for this introduction would be by using a so-called "alternating oracle turing machine" where every oracle call can be labelled with quantifier $\exists$ or $\forall$ (for layer i with $i-1$ quantifier changes allowed).

Theorem 2.10. If there is any $i \geq 1$, $\Sigma_i P = \Pi_i P$, then $\forall j > 1$, $\Sigma_j P = \Pi_j P = \delta_j P$.

Remark. For $i = 1$ we can conclude that (under the assumption that $P = NP$) the layer 1 already does not tell us anything and the whole hierarchy definition is useless ("collapse at layer 1"). If the assumption does not hold we still do not know whether or not the hierarchy collapses at any finite layer.

Conjecture. P $\neq$ NP and polynomial hierarchy does not collapse for any finite layer.

Proof. It suffices

$$\Sigma_i P = \Pi_i P$$

$$\Rightarrow \Sigma_{i+1} P = \Pi_{i+1} P$$

Consider $L \in \Sigma_{i+1} P \overset{\text{Theorem 2.7}}{\Rightarrow} \exists$ relation R in $\Pi_i P$ ($= \Sigma_i P$ according to assumption) with $L = \{x : \exists y : (x, y) \in R\}$. R is also in $\Sigma_i P \Rightarrow (x, y) \in R \Leftrightarrow \exists z$ with $(x, y, z) \in S$ for any relation S of $\Pi_{i-1} P$.

Thus $x \in L \Leftrightarrow \exists$ a string $y; z$ such that $(x, y, z) \in S$ with $S \in \Pi_{i-1} P \Rightarrow L \in \Sigma_i P$ (analogously for $\Pi_i P$).

10.3 QSAT$_i$

Consider a quantified SAT problem QSAT$_i$ with i quantifiers.

Given. A SAT equation ϕ. The variables in ϕ are partitioned in i classes $X_1, X_2, \ldots, X_i$.
Question. Is it true, that a partial truth assignment for X_1 exists such that for all partial truth assignment of variables in X_2 there exists partial truth assignments for variables in X_3 such that $\ldots X_1$.
Problem can analogously defined to start with $\forall$ quantifier. $\exists X_1 \forall X_2 \exists X_3 \ldots \phi$

Theorem 2.11. QSAT$_i$ is $\Sigma_i P$-complete.

Remark: The variant with an $\forall$ quantifier at the start is $\Pi_i P$ complete.

This is an example at the i-th layer. We now know that at the i-th layer complete problems do exist.

Date. 28th of Oct 2013

Proof.

1. QSAT $\in \Sigma_\mathrm{i}\mathrm{P}$ follows directly from the non-recursive representation of $\Sigma_i P$.
2. We show $\Sigma_i P$-hardness. Consider $L \in \Sigma_i P$. Find a reduction of L to QSAT: Transform L to the non-recursive characteristic of $\Sigma_i P$ (corollary 2.9). Relation R can be decided in polynomial time. $\Rightarrow \exists$ polynomial turingmachine M which accepts all input strings $(x_i; y_1; \ldots; y_i)$ such that $(x; y_1; \ldots; y_i) \in R$.

Remark. (for $\Sigma_i P$-hardness) If i is odd (i even analogously) we use the method introduced in proof of Cook's Theorem. $\exists$ SAT equation ϕ which reflects the computation of M. We arrange the variables in $i+2$ variables. The set X corresponds to the variables in the input string of the form $(x_i; y_1; \ldots; y_i)$ as mentioned previously (each set is separated by a semicolon). Show: All variables that describe the resulting components of M.

For a fixed assignment of the variables $X, Y_1, \ldots, Y_i$ we state: The resulting boolean expression is satisfiable $\Leftrightarrow$ the variables of the input string describe a string accepted by M.

x is a random string of appropriate length and substitutes in ϕ the corresponding assignment for the variable X.

We know:

$$x \in L \Leftrightarrow \underbrace{\exists y_1 \forall y_2, \ldots, \exists y_i}_{\text{i is odd}} \text{ such that } R(x, y_1, \ldots, y_n)$$

Thus for the partial assignment of the variable X with x there are values in Y_1 such that for all values of variables in Y_2 ... there are values for variables in Y_i and there are values for the variable in Z such that ϕ is satisfied.

Another example. Multilevel-Optimization is $\Sigma_i P$-complete.

Theorem 2.12. If there is a PH-complete problem, the polynomial hierarchy collapses at finite level.

Proof. Assume L is PH-complete. Thus there exists a $i \geq 0 : L \in \Sigma_i P$. Each language $L' \in \Sigma_{i+1} P$ can be reduced to L. Because all levels of PH are closed under reductions, we conclude $L' \in \Sigma_i P \Rightarrow \Sigma_{i+1} P = \Sigma_i P$. Thus the polynomial hierarchy collapses.

Remark: It's an open question whether PH-complete problems exist (conjecture: no).

Theorem 2.13: PH $\subseteq$ PSPACE (conjecture: PH $\subset$ PSPACE).

Remarks:

1. Quantified boolean equation problem.
$$Q_1x_1Q_2x_2\dots Q_kx_k\underbrace{\phi(x_1,\dots,x_k)}_{\text{quantifier-free}}\quad Q_i\in\{\forall,\exists\}\,.$$
is PSPACE-complete.
2. If we define Σ_iP, Π_iP with an alternating oracle-TM, we have to assume that the number i of quantifiers is not part of the input. This is a constraint for the number of allowed quantifiers. There is no constraint for PSPACE.
3. Because there are PSPACE-complete problems, we can conclude from PSPACE = PH that the polynomial hierarchy collapses at finite level.

10.4 $BPP\subseteq\Sigma_2P$

Theorem 2.14. $BPP\subseteq\Sigma_2P$

Proof. $L\in$ BPP. So there exists a turingmachine M with computations of length $p(n)$ (p polynomial) for inputs of length n such that M decides L using majority vote (probability for false negatives is for example $\leq\frac{1}{4}$). For each input x of length n denote $A(x)\in\{0,1\}^{p(n)}$ the set of accepted computations.

We can assume that

$$\forall x\in L:|A(x)|\geq 2^{p(n)}(1-\frac{1}{2^n})\quad\text{error probability}\leq\frac{1}{2^n}$$
$$\forall x\notin L:|A(x)|\leq\frac{1}{2^n}\cdot 2^{p(n)}\quad\text{error probability}^3\leq\frac{1}{2^n}$$

Denote U the set of bit strings of size $p(n)$.

Definition. $a,b\in U$ with $a\oplus b$ as component-wise XOR operation.

Observation.

1. $a\oplus b=c\Leftrightarrow c\oplus b=a$
2. $(a\oplus b)\oplus b=a$. The function $\oplus b$ is reversible with the same argument.
3. $a\in U$ is constant, $r\in U$ is random value. So $a\oplus r$ is random value.

[3]can easily be achieved by repetition

Definition. (Translation) $t \in U$. $A(x) \oplus t := \{a \oplus t | \, a \in A(x)\}$ (translation of A with t).

From the second property we can conclude that $|A(x) \oplus t| = |A(x)| \forall t \in U$.

We will show that we can return for $x \in L$ a relatively small number of translations which cover U (cardinality $p(n)$) whereas for every $x \notin L$ there is no such representation.

$x \in L$. $t_1, t_2, \ldots, t_{p(n)} \in U$ is a random sequence of $p(n)$ translations ($p(n)^2$ coin throws).

$b \in U$. b is "cover by" $t_1, \ldots, t_{p(n)}$ if $b \in A(x) \oplus t_j$ for any $j \in \{1, \ldots, p(n)\}$.

Question: What's the probability that b is covered?

$$b \in A(x) \oplus t_y \overset{\text{observation}}{\Leftrightarrow} \qquad \underbrace{b \oplus t_j}_{\text{randomized because of third property}} \qquad \in A(x)$$

It has the same distribution like $t_1, \ldots, t_n$.

Because $x \in L \Rightarrow$ probability$(b \notin A(x) \oplus t_i) \leq \frac{1}{2^n} = 2^{-n}$.

- Probability that b is not covered by any t_j is $\leq 2^{-n \cdot p(n)}$.
- Every element of U is not covered with probability $\leq 2^{-n \cdot p(n)}$.
  - Probability that some element of U is not covered:
    $$\leq 2^{-np(n)} \cdot \underbrace{|U|}_{2^{p(n)}} = 2^{-(n-1) \cdot p(n)} \ll 1$$

A sequence of $p(n)$ random translations covers the complete U with correspondingly high probability.

In the case $x \notin L$ $A(x)$ is an exponentially small portion of U. Thus for any sufficiently large n there is no sequence of $p(n)$ translations that cover U completely.

Conclusion. $x \in L \Leftrightarrow$ there exists a sequence of polynomial ($p(n)$) number of translations that cover U.

$$L = \big\{x : \exists T = (t_1, t_2, \ldots, t_{p(n)})\big\} \quad [t_j \text{ translations}]$$
such that $\forall b \in U \exists j \in \{1, \ldots, p(n)\} : b \oplus t_j \in A(x)$.

The last exists operator can be rewritten as: $(b \oplus t_1 \in A(x)) \vee (b \oplus t_2 \in A(x)) \vee \ldots \vee (b \oplus t_{p(n)} \in A(x))$ (can be tested in polynomial time). Therefore we get a Σ_2 p-characterisation of L; thus $L \in \Sigma_2 p$.

We know that BPP^{i-1} is closed under complement $\Rightarrow$ BPP $\subseteq \Sigma_2 P \cap \Pi_2 P$.

11 3 counting problems

Not combinatorics, but relations (specially in graph theoretical) problems. From a complexity theoretical point of view: How many solutions are there for a given problem?

11.1 #SAT

Given a SAT equation, find the number of satisfying truth assignments. Analogously the number of hamiltonian paths - number of hamiltonian paths. Analogously the number of cliques - number of cliques with $\geq k$ vertices.

Those examples are couting variables for problems where the basic decision problem is NP-complete.

11.2 #MATCHING / PERMANENT

Given a graph $G = (V \cup U, E)$ with $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_n\}$. Find the number of perfect matchings in G (this corresponds to a 1-by-1 mapping of vertices in U to vertices in V).

We consider the adjacency matrix A^G which is associated to G. $A(G) = (a_{ij})$ with

$$a_{ij} = \begin{cases} 1 & \text{if } \{u_i, v_i\} \in E \\ 0 & \text{else} \end{cases}$$

When does the association π (permutation $\pi \in S_n$) describe a perfect matching?

$$a_{i\pi(i)} = 1 \forall i \in \{1, \dots, n\} \Leftrightarrow \left\{u_i, v_{\pi(i)} \in E \forall i \in \{1, \dots, n\}\right\} \Leftrightarrow \prod_{i=1} a_{i\pi(i)} = 1$$

The number of perfect matchings is given by

$$\operatorname{perm}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i\pi(i)}$$

$\pi \in S_n$ is the set of permutation of $\{1, \dots, n\}$. Compare

$$\det A(G) = \sum_{\pi \in S_n} (-1)^{\operatorname{sign}(\pi)} \cdot \prod_{i=1}^{n} a_{i\pi(i)}$$

Question. There exists an efficient algorithm for computation of $\operatorname{perm}(A)$ or at least for 01-matrices (restriction of values)?

11.3 Subgraph paths

Given a graph with m edges and n vertices. Find the number of subgraphs of G (2^m at maximum) which contain a path from 1 to n.

12 #P

Spoken "number P" or "sharp P". Is the class of counting problems that are associated using relation Q. All previously mentioned problems are part of #P.

Q is a polynomial balanced in polynomial time decidable relation. The associated counting problem for Q: Given an x, find the number of y with $(x, y) \in Q$?

Question. Are there any of these problems #P-complete? (be aware an appropriate reduction term is necessary) #SAT $\in$ #P-complete?

Date. 4th of Nov 2013

For the definition of $\#P$-completeness we have two approaches:

- by using the reduction term, we defined for function problems (number of solutions corresponds to solution). We need an S which gives us the number of solutions for B with the number of solutions for A. This subclass of reductions is popular for counting problems: "paisimonious reductions" (dt. ,,sparsam") (number of solutions is preserved or with a factor k, $k \in \mathbb{N}$).
- Informally:

  > A function problem f is $\#P$-complete iff problem is in $\#P$ and the existence of a polynomial algorithm for evaluation of f implies $\#P = FP$.

  We extend the concept of oracle TMs to simple function languages (for oracle TMs: access to an oracle $L \subseteq \{0,1\}^*$ and we can answer questions like $q \in L$? in one oracle call). Now we provide a function $f : \{0,1\}^* \to \{0,1\}^*$ to the turingmachine M (including an oracle to decide f). ($f : \{0,1\}^* \to \mathbb{N}$ can be applied to binary representation). Thus m has no access to the language $L = \{(x, i)$ with $f_i(x) = 1\}$ with i as bit index. For function $f : \{0,1\}^* \to \{0,1\}^*$ FP^f denotes the set of functions which can be decided with a polynomial turingmachine with oracle calls to f.
  Formally: Function f is $\#P$-complete iff $f \in \#P$ and $\forall g \in \#P$ we conclude $g \in \mathrm{FP}^f$. Obviously $f \in FP \Rightarrow \mathrm{FP}^f = FP$.

Theorem 3.1. *f is #P-complete and $f \in$ FP. Thus FP = #P.*

Theorem 3.2. *#SAT is #P-complete.*

Sketch of proof. Consider the proof of Cook's theorem. Reduction of an arbitrary language to SAT. In polynomial time evaluable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $\forall x \in \{0,1\}^* : f(x) \in \text{SAT} \Leftrightarrow \text{x} \in \text{L}$.

This proof however provides more than that: It provides a method to convert a certificate $x \in L$ to a certificate $f(x) \in \text{SAT}$ [ie. a satisfying truth assignment] and vice versa. This is a bijunctive relation.

Thus the number of satisfying assignments is the number of certificates for x (the reduction is a paisimonious reduction).

Theorem 3.3. $\#\text{HAMILTONIAN} - \text{PATHCYCLE} \in \#\text{P-complete}$.

Reduction via $\#SAT$ (can be dervied from the proof that TSP is FP^{NP} complete).

Question. Are there any $\#P$-complete counting problems whose base problem (as decision problem) lie in P?

Answer. Yes. Two examples:

- PERMANENT is $\#P$-complete (see theorem 3.4).
- #CYCLE (given a graph, count number of cycles). There exists a polynomial algorithm for #CYCLE $\Rightarrow$ P = NP.

Theorem 3.4. (Valiant, 1979) PERMANENT $\in$ #P-complete.

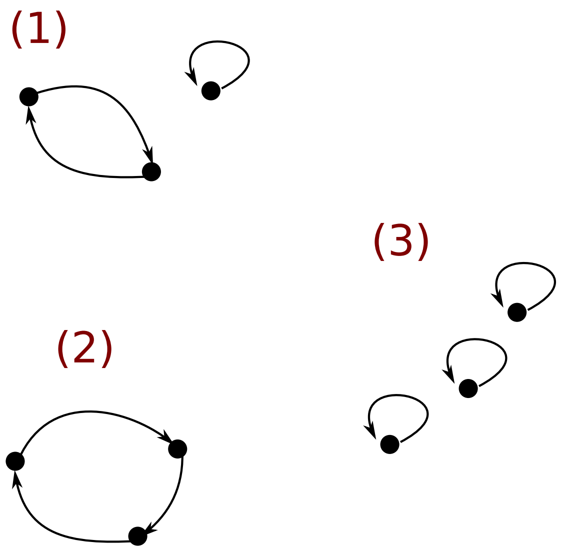
Proof. Goal: perm(A) evaluation for 01 matrices. We use a proof using general matrices (integers as entries of the matrix).

1. For 01-matrix A we already know perm(A) = the number of perfect matchings in a bipartite graph $G(A)$ which are associated to A.
2. For matrix A with $a_{i,j} \in \{0, \mp 1\}$. It is easy to state that perm(A) = $|\left\{\pi \in S_n : \pi_{j=1}^{n} a_{i\pi(i)} = 1\right\}| - |\left\{\pi \in S_n : \pi_{i=1}^{n} a_{i\pi(i)} = -1\right\}|$. Two #SAT call are sufficient.
3. A is a $n \times n$ matrix with integers as entries. We can consider A as weighted adjacency matrix of the directed, complete[4] graph (rows) with $V = \{v_1, \ldots, v_n\}$ (columns) and the following vertices set: The edge (v_i, v_j) has weight $a_{i,j}$. Loops are allowed (and are contained).

Observation. Every permutation $\pi \in S$ corresponds to in $G(A)$.

So-called cycle cover. Each component is a directed cycle and each vertex occurs in exactly one component.

[4]complete, because we assign weight 0 to missing edges



Define the weight of one cycle cover as the product of the weights of contained edges: perm(A) is the sume of weights of all cycle covers.

Remark. This way we can show that PERMANENT $\in \mathrm{FP}^{\#\mathrm{SAT}}$ (for arbitrary integer matrices).

In the following we will omit edges with weight 0 in our drawings (cycle covers containing such edges have weight 0; therefore we can ignore it). In the construction we will also allow multiple edges.

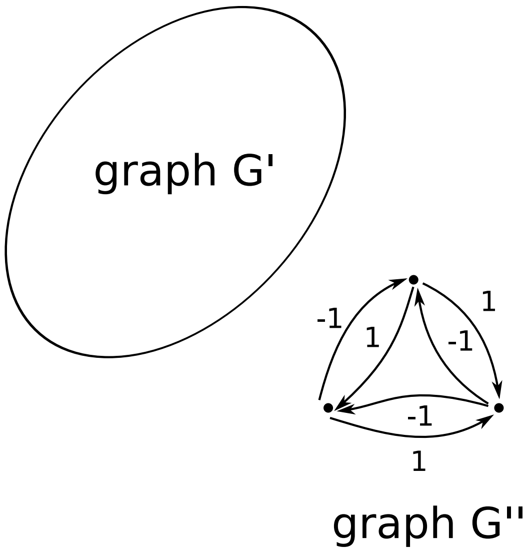
Observation. (cycle cover observation) Furthermore we observe: G consists of G' and G'' (G' can be arbitrary). G'' has 2 cycle covers with weight $\neq 0$ (be aware that self-loops are not drawn).

Each cycle cover of G consists of one cycle cover of G' and one cycle cover of G''. Therefore we have two possibilities: one with 1 and one with -1 (with weight $\neq p$). For cycle cover with weight ω for G' we get the terms ω and $-\omega$. The overall weight sum of all cycle covers is 0.

Now we reduce #3SAT (which is #P-complete) to PERMANENT.

Now a 3SAT equation ϕ with n variables and m clauses is given. We will construct an integer matrix A or an equivalent directed, weighted graph $G(A)$ (let's call him $\tilde{G}$) (negative entries will be used) such that $\mathrm{perm}(A) = \mathrm{perm}(\tilde{G}) = 4^{3m}$ ($\#\phi$; which is the number of satisfying assignments of ϕ). $\tilde{\mathrm{G}} \Rightarrow \mathrm{P}(A)$.

Later on $\tilde{G} \rightarrow \hat{G}$ with 0-1 weights such that $\mathrm{perm}(\tilde{G})$ evaluates $\mathrm{perm}(\hat{G})$ (then



we have shown that 01-problems are $\#P$-complete).

The central idea is that there are two types of cycle covers in $\tilde{G}$. The ones that are assigned to a satisfying assignment of ϕ and the other ones. Similar to the cycle cover observation we will use negative edge weights to achieve that the values of cycle covers (which do not contribute to the satisfying assignment) drop out each other.

Furthermore every cycle cover contributes 4^{3m} to each satisfying assignment for $\mathrm{perm}(G)$.

$$\mathrm{perm}(\hat{G}) = \mathrm{perm}(A) = 4^{3m}$$

($\#\phi$ as considered).

Question: ϕ reduces to $\tilde{G}$? There are 3 kinds of gadgets:

- clause-gadgets
- variable-gadgets
- consistency-gadgets (XOR-gadgets)

Consider the following 4×4 matrix or the corresponding graph

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

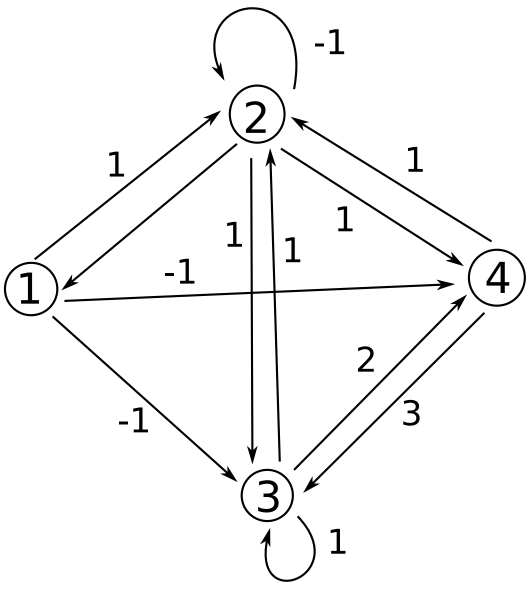


Figure 1: Example graph

We can now explicitly evaluate that

- perm$(A) = 0$
- For perm(B): We can dervice B from A by deleting the first row and first column $\Rightarrow$ perm$(B) = 0$. Or we could also delete the fourth row and fourth column. Or we could also delete the first and fourth row and column. Also for matrix it results if first and fourth row and first and fourth column is deleted.
- perm$(C) = 4$ where B is created by deleting of the first row and fourth column or the fourth row and first column of matrix A.

The upper graph in the illustration can be derived from the upper graph and the lower graph represents the remaining graph.

For the modified example graph with g we state: The overall weight of all cycle covers is 8 (contribution value 4 results from cycle cover with $(g, 4)$ and $(1, g)$ -

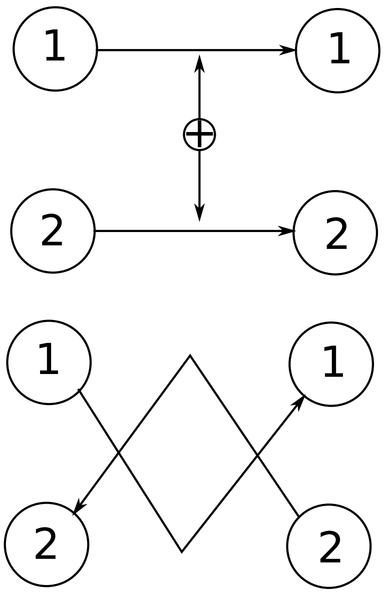


Figure 2: Schematic representation

this corresponds to deletion of the first row and fourth column and the second value 4 results from $(g, 1)$ and $(4, g)$ correspondingly.)

The contribution of the cycle cover which contains g is 0 (corresponds to matrix A without deleting rows and columns). Also: For all cycle covers that traverse $(g, 1)$ and $(1, g)$ (deleting first row and first column) we get contribution value 0. Correspondingly for $(g, 4)$ and $(4, g)$ for the fourth row and column.

We can use this to rewrite XOR: Assume that we have a graph H which contains the two edges $(1, 1')$ and $(2, 2')$.

Consider the lower graph in the schematic drawing. In combination with the definitions from above: The sum of all weights, the cycle cover of H which traverses $(1, 1')$, but does not traverse $(2, 2')$. All other cycle covers result in value 0.

Regarding clause gadgets: The traversal of an external edge in one cycle cover corresponds to one not-satisfied corresponding variable. There are three external edges, one per variable.

In each clause we have 3 variables which are represented by the two external edges of this clause. These external edges are connected to the edges of the corresponding variable gadgets via XOR gadgets (which is equivalent to our construction of for the hamiltonian path problem).

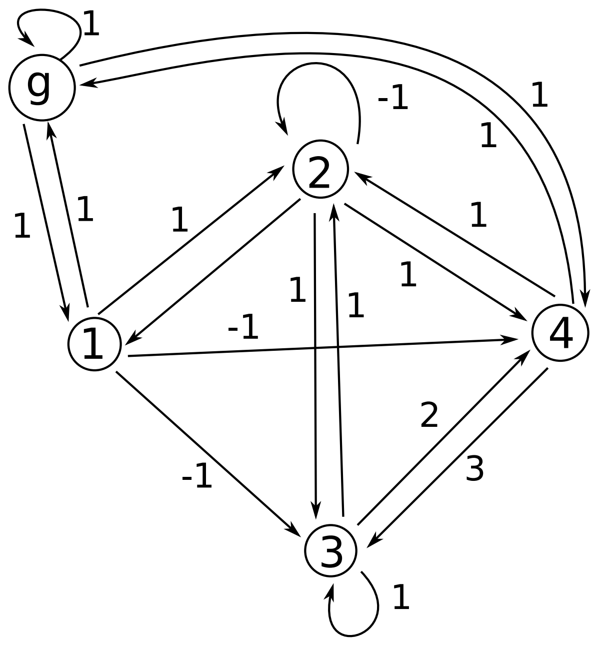


Figure 3: Example graph with g added

Observation. There does not exist a cycle cover which goes through all 3 external edges. Furthermore for all proper subsets of the set of 3 external edges (including the empty set): There exists 1 cycle cover which contains exactly all those edges and no other ones.

With this specific setup we achieve that the remaining graph G' satisfies:

$$\mathrm{perm}(G') = 4^{3m}(\#\phi)$$

with $\#\phi$ has the number of satisfying assignments for ϕ.

Remark. The corresponding external edges are omitted for clauses which contain x_i positively and added for clauses containing x_i negatively.

12.0.1 Reduction to 0-1 matrices

Computed in 2 steps:

1. Construct a graph G'' with edge weights $\in \{0, \pm 1\}$ and $\mathrm{perm}(G'') = \mathrm{perm}(G')$. Edge weights which are powers of 2 (2^k) can be represented by a path of k edges of weight 2 (all new inner edges of this path do only occur in this path). For an edge of weight $2^k + 2^{k'}$ we construct to two parallel paths with weights 2^k and $2^{k'}$. Each edge whose weight is not a

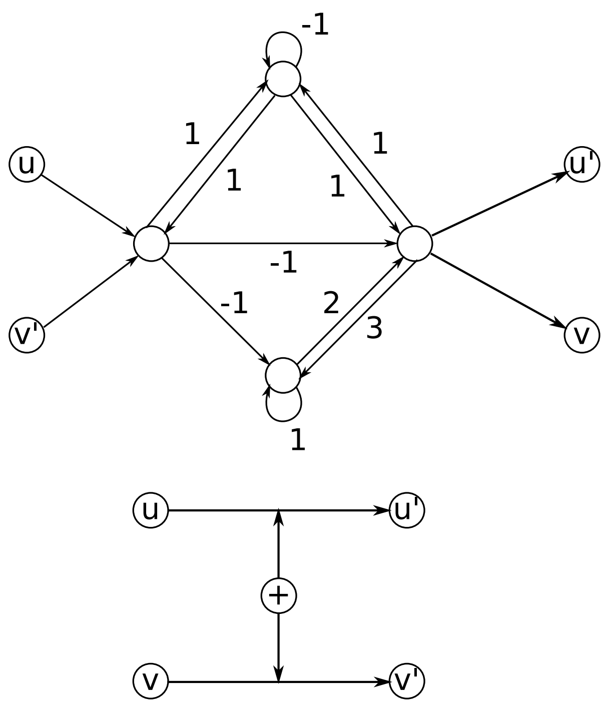


Figure 4: XOR gadget (upper drawing) and its schema (lower drawing)

power of 2 can be represented as a set of parallel paths which use binary representation. If L is the number of bits to describe the weights in G', the graph is blown back with factor $\mathcal{O}(n \cdot L^2 \log n)$. 2 can be replaced by parallel 1-edges. Therefore we get only $\{0, \mp 1\}$ edges.

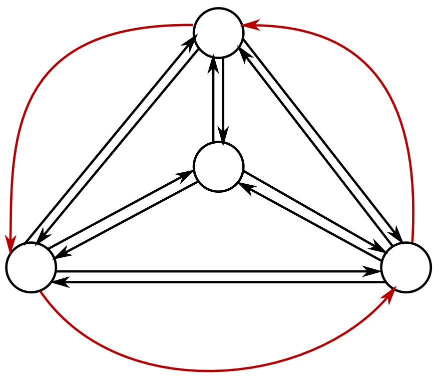
2. Removal of negative weights by transition to modulo arithmetics. Transition to $\mod 2^q + 1$ with $q = n^2$. Now we can state $-1 \equiv 2^q \mod 2^q + 1$. The permanent $\mod 2^q + 1$ keeps the same, if edges with weight -1 get replaced by edges with weight 2^q. Those get replaced by a subgraph with all weights 1 edges (size $\mathcal{O}(q) = \mathcal{O}(n \log n)$). Graph G''' with weights $0, 1$ such that the permanent of G''' is used to evaluate the original permanent (rest $\mod 2^q + 1$). Blow back with factor $\mathcal{O}(< n \log n)$.

Remark. The exact counting of solutions is sometimes computationally expensive. Let's do approximations!

α-Approximation

$$0 < \alpha < 1$$

A provides an α-approximation for $f : \{0,1\}^* \to N$ if $f(x) \leq A(x) \leq \frac{f(x)}{\alpha} \forall x$.



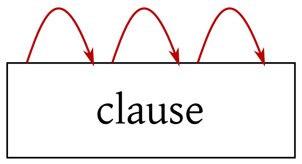


Figure 5: Clause gadget (upper drawing) and its schema (lower drawing). Red edges are external ones.

There are counting problems for which even an approximative algorithm for constant $\alpha > 0$ is difficult. But there are also problems which are easy to address approximatively.

For the (01) PERMANENT problem: There exists a FPRAS (fully polynomial randomized approximation scheme) algorithm which (for a given ε and δ) evaluates a $(1-\varepsilon)$ approximation for requested function $f : \{0,1\}^* \to \mathbb{N}$ (for 01 perm = number of perfect matchings) with probability $1-\delta$ (with probability δ the algorithm is allowed to be faulty) in time $\underbrace{p(n, \log\frac{1}{\delta}, \log\frac{1}{\varepsilon})}_{\text{polynomial}}$.

Remark. In case N = NP, then for all problems in #P a FPRAS[5] can be found. Even a FPTAS[6] (see last chapter of this lecture).

[5]randomized

[6]deterministic

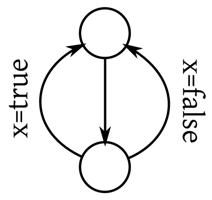


Figure 6: A variable gadget (2 possible cycle covers)

12.0.2 Classification of #P: How powerful is counting?

Problems in #P are solvable with polynomial limited space (eg. lexiographical enumeration). Therefore #P is *not* more powerful than PSPACE. PH is not more powerful than PSPACE.

What's the relation of $\underbrace{\#\text{P}}_{\text{class of functions}}$ and $\underbrace{\text{PH}}_{\text{class of languages}}$?

Theorem 3.5. (Theorem of Toda) $\text{PH} \subseteq \text{P}^{\#\text{P}}$ and $\text{PH} \subseteq \text{P}^{\#\text{SAT}}$.

Therefore we can solve every problem in the polynomial hierarchy if an oracle for a #P-complete problem is provided.

Remark. $\#\text{P} = \text{FP} \Rightarrow \text{NP} = \text{P} \wedge \text{PH} = \text{P}$.

Conclusion. Informally: Counting is more powerful than the polynomial hierarchy.

Relation between PP and #P? So the question of PP is whether or not half of the evaluations result in acceptance. Thus we are only interested in the MSB of the number of accepted evaluations. #P is interested in the values of all bits.

Idea. We are only interested in the LSB (parity). This defines the complexity class $\oplus P$ ("parity P", "odd P").

A language L is in ⊕P if there is a polynomial turingmachine M such that for all strings x we can state that:

$$x \in L \Leftrightarrow \text{number of accepting evaluations of x at M is odd}$$

(equivalently there exists a polynomially balanced relation R decidable in polynomial time such that $x \in L \Leftrightarrow$ the number of y with $(x, y) \in \mathbb{R}$ is odd).

12.0.3 Problem: $\oplus$SAT

Given a SAT equation. The question is, is the number of satisfying assignments even?

Theorem 3.6. $\oplus$SAT is $\oplus P$ complete.

Proof. $\oplus$SAT $\in \oplus$P according to definition. $\oplus$P completeness follows from parsimonious reducibility of a problem in #P to #SAT.

Analogously, $\oplus$HAMIL.PATH/CYCLE $\in \oplus$P-complete.

12.0.4 $\oplus$PERM in $\oplus$P-complete

Can be solved in polynomial time because the parity of the determinant corresponds to the parity of the permanent (polynomial time!).

Theorem 3.7. $\oplus P$ a.g. regarding complement. Proof by Ü.

Theorem 3.8 NP $\subseteq$ RP$^{\oplus \mathrm{P}}$. We will define a polynomial Monte Carlo algorithm for SAT when using a $\oplus$SAT oracle.

12.1 NP $\subseteq$ RP$^{\oplus \mathrm{P}}$

Proof. We will define a polynomial Monte-Carlo algorithm for SAT which has access to a $\oplus$ SAT oracle. Given a SAT equation ϕ with variabes $x_1, \ldots, x_n$.

Is $S \subseteq \{1, \ldots, n\}$.

Definition. (hyperplane η_s) A hyperplane η_s is a boolean expression requiring that an odd number of variables with indices of S have value true.

$y_0, \ldots, y_n$ are new variables. η_s can be constructed with the following set of rules:

- Add (y_0) as clause.
- Add (y_n) as clause.
- For all $i \in \{1, \ldots, n\}$ add clause $\tilde{C}_i$

$$\tilde{C}_i = \begin{cases} (y_{i-1}) \oplus x_i & i \in S \\ (y_{i-1}) & i \notin S \end{cases}$$

(Transform this into a DNF.)

Is $\phi_0 = \phi$ the whole SAT equation. For $i = 1$ to n repeat the following steps:

1. Create a random set $S_i \subseteq \{1, \dots, n\}$ and set

$$\phi_i = \phi_{i-1} \wedge \eta_{s_i}$$

Apply $\oplus$-SAT oracle to ϕ_i. If $\phi_i \in \oplus$ SAT, then ϕ satisfiable. Return "YES". If $\phi_i \notin \oplus$ SAT $\forall j = 1, \dots, n$, return "No or probably no".

Claim. The algorithm above is a Monte-Carlo algorithm.

Proof.

- No false positive answers.

$$\phi_i \in \oplus\text{SAT} \Rightarrow \phi_i \text{ has } \geq 1 \text{ satisfying assignment}$$

$$\Rightarrow \phi \text{ has more than 1 satisfying assignment}$$

- Probability for false negatives? Claim: $\leq \frac{7}{8}$ (6 repetitions required for probability $\leq \frac{1}{2}$).

  Observation. If the number of satisfying assignments for ϕ is between 2^k and 2^{k+1} for $0 \leq k < n$, then ϕ_{k+2} has exactly one satisfying assignment with probability $\geq \frac{1}{8}$.

  Proof. T is the set of satisfying assignments for ϕ. $2^k \leq |T| \leq 2^{k+1}$. We define that two truth assignments in hyper planes η_s do correspond if both satisfy η_s or both do not satisfy η_s.

  Let's fixate $t \in T$. Consider $\hat{t} \in T$. The probability that t corresponds with t' at the first $k+2$ hyper planes $(\eta_{s_1}, \eta_{s_2}, \dots, \eta_{s_{k+2}})$ is $\frac{1}{2^{k+2}}$ (which results from the fact that all $(k+2)$ sets $s_1, \dots, s_{k+2}$ contain an even number of variables where t and $\hat{t}$ do not correspond and these events are indepedent and occur with probability $\frac{1}{2}$).

  Build sum over $\hat{t} \in T \setminus \{t\}$. The probability that t corresponds to some $\hat{t} \in T \setminus \{t\}$ at the first $k+2$ hyper planes is

$$\leq \frac{|T| - 1}{2^{k+2}} < \frac{1}{2}$$

  Opposite probability (they do not correspond) is

$$\geq \frac{1}{2}$$

  Obviously we conclude that t satisfies the first $k+2$ hyperplanes with probability $\frac{1}{2^{k+2}}$.

  We now state: If t satisfies the first $k+2$ hyper planes then t is the only assignment of T with this property with probability $\geq \frac{1}{2}$ (the probability that t satisfies the first $k+2$ hyper planes does not interfere with the probability that $\hat{t} \neq t$ do not correspond).

With probability $\geq \frac{1}{2^{k+3}}$ $(=\frac{1}{2}\frac{1}{2^{k+2}})$ t is the only satisfying assignment for ϕ_{k+2}.

This is valid $\forall t \in T$. Because of $|T| \geq 2^k$ the probability that such $t \in T$ exists is

$$\geq \underbrace{2^k}_{\text{lower bound for } |T|} \cdot \frac{1}{2^{k+3}} = \frac{1}{8}$$

Our observation is proven.

If ϕ is satisfiable, then $\exists k \in \{0, \ldots, n-1\}$ with number of satisfying assignments for ϕ between 2^k and 2^{k+1}. Therefore at least one of the equations ϕ_i has exactly one satisfying assignment with probability $\geq \frac{1}{8}$ and thus an odd number and thus $\phi_i \in \oplus$ SAT.

Our claim is proven.

13 Interactive protocols

13.1 Introduction and Terminology

In math the classical concept of proofs is strongly related to the idea of NP certificates. But the verification is done without any interaction. We want to introduce 2 roles/actors:

- Prover
- Verifier

A prover creates proofs and the verifier checks those proofs. During this process they can interact by exchanging messages. At the end the verifier has to decide whether or not the proof gets accepted.

Open question. Which power do prover and verifier have?

13.1.1 Variant 1: Prover and verifier act deterministically

Definition. (Message exchange) $f, g : \{0,1\}^* \to \{0,1\}^*, k \in \mathbb{N}_0$ (can depend on input length). k rounds of interaction is a result of strings $a_1, \ldots, a_k \in \{0,1\}^*$.

$$a_1 = f(x)$$

$$a_2 = g(x, a_1)$$

$$\vdots$$

$$a_{2i+1} = f(x, a_1, \ldots, a_{2i}) \qquad 2i < k$$

$$a_{2i+2} = g(x, a_1, \dots, a_{2i+1}) \qquad 2i + 1 < k$$

In the end the verifier decides whether or not to accept the proof. This decision only depends on x and $a_i \Rightarrow \{0, 1\}$ for acceptance or rejection.

Requirement for a deterministic proof system. The language L has a deterministic proof system with k rounds if there is a deterministic turing machine M which satisfies the following constraint for an input $x, a_1, a_2, \dots$ in polynomial time in $|x|$ and with k message exchanges:

$$x \in L \Rightarrow \exists \underbrace{P}_{\text{proof}} : \{0,1\}^* \to \{0,1\}^*$$

$$\text{with propability(verifier accepts } P) = 1$$

$$x \notin L \Rightarrow \forall \underbrace{P}_{\text{proof}} : \{0,1\}^* \to \{0,1\}^*$$

$$\text{with propability(verifier does not accept } P) = 0$$

Complexity class $d|P$ (“d” for deterministic) contains all languages for which a deterministic proof system with poly(n) rounds can be found.

Lemma 4.1. $d|P = NP$. (Proof is given in practicals.)

Consequence: With a purely deterministic approach we don’t gain any new results. Interaction is not relevant.

13.1.2 Variant 2

More interestingly, we consider a randomized verifier. Verifier can return false answer, but we add constraints for the probability.

Power of verifier. Polynomial randomized algorithm.

Power of prover. Exponential runtime. Deterministic is enough (or alternatively PSPACE).

13.2 Example Scenario

The input x is known to both actors. Bob (prover) and Veronica (verifier) act alternating. Bob runs algorithm $\tilde{B}$ and Alice runs algorithm $\tilde{V}$. They exchange messages $m_1, m_2, \dots$ of polynomial size in $|x|$.

Assume we start with Bob (could also be Veronica).

$$m_1 = \tilde{B}(x) \quad \text{first message of Bob, depending on } x$$

$$\vdots$$

$m_{2i} = \tilde{V}(x, m_1, \ldots, m_{2i-1}, V_i)$ message by Veronica with V_i as private random bits

$$m_{2i-1} = \tilde{B}(x, m_1, \ldots, m_{2i-2})$$

The protocol ends with Veronica's message "YES" (accept) or "No" (reject). The tuple (B, V) decides a language L for all inputs iff

- if $x \in L$, then the probability that x gets accepted by (B, V) is greater equal some constant between 0 and 1 (eg. $\geq \frac{3}{4}$).
- if $x \notin L$, then the probability that x gets *accepted* by (B', V) (with B' as any algorithm with exponential runtime (or PSPACE algorithm)) is $\leq \frac{1}{4}$ (or any other constant).

Remark. We can again use repeated runs to reduce the error probability.

Definition. (complexity class IP) The class of languages which can be decided with an interactive protocol (of the previously described manner) with polynomial rounds.

Definition. (complexity class IP(l)) The number of rounds is restricted to l.

Obviously

$$\text{NP} \subseteq \text{IP} \qquad \text{(no randomization necessary)}$$

$$\text{BPP} \subseteq \text{IP} \qquad \text{(verifier does not require prover)}$$

Result of Shamir.

$$\text{IP} = \text{PSPACE}$$

Remark. In our definition of IP we had

$$x \in L \Rightarrow \exists \text{ proof } P : \text{probability(V accepts P)} \geq \frac{3}{4}$$

$$x \notin L \Rightarrow \forall \text{ proof } P : \text{probability(V rejects P)} \leq \frac{1}{4}$$

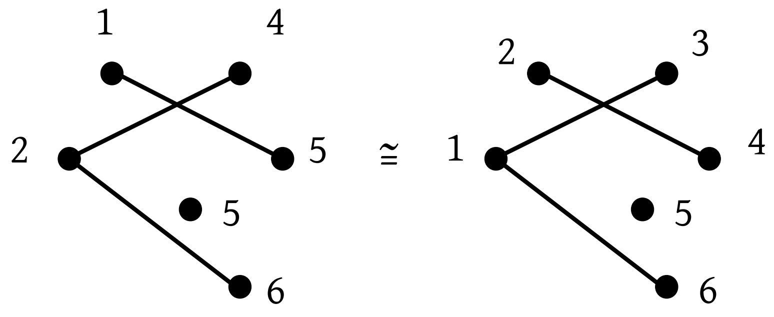
- We could also replace $\geq \frac{3}{4}$ with $= 1$ without modifying complexity class IP (non-trivial result).
- But $\leq \frac{1}{4}$ cannot be replaced by $= 0$ without falling back to NP.
- In definition of IP the verifier uses private random bits. The constraint to public random bits leads to Arthur-Merlin proofs (AM).

13.3 Graph isomorphism (GI) or complement and interactive protocols

Given. 2 graphs $G_0 = (V_0, E_0), G_1 = (V_1, E_1)$ with $|V_0| = |V_1|$ and $|E_0| = |E_1|$.
Question. Are G_0 and G_1 isomorphic ($G_0 \cong G_1$)?

13.4 Complement of graph isomorphism

Given. (like in previous graphs)
Question. Are G_0 and G_1 not isomorph?



$$\text{GI} \in \text{NP}$$
$$\Rightarrow \text{GI} \in \text{IP}(1) \subseteq \text{IP}$$

What about $\overline{\text{GI}}$?

Complexity state of GI is an open question. We don't know know whether GI is NP-complete. Experts assume No.

Theorem 4.2. (without proof) $\overline{\text{GI}} \in \text{IP}(2)$ also $\overline{\text{GI}} \in \text{IP}$.

Proof. Construct with an appropriate protocol:

Verifier: Select random bit $b \in \{0,1\}$. Create random permutation $\pi \in S_n$ and determine graph $H = \pi(G_b)$. Verifier sends H to Prover.

Prover: Prover's goal is to determine whether or not which graph (G_0 or G_1) was permuted by verifier. Determine bit $\tilde{b} \in \{0,1\}$ and send it to Verifier. (If $G_0 \cong G_1$, Prover chooses bit $\tilde{b}$ randomly and otherwise select $\tilde{b}$ in such a way that $H \cong \pi(G_{\tilde{b}})$).

Verifier: Accepts, if $b = \tilde{b}$ and rejects for $b \neq \tilde{b}$.

Communication protocol with 2 rounds: Essential part of protocol: Random bits of Verifier are private.

To prove that this is an IP protocol, consider the following 2 cases:

1. $G_0 \cong G_1$: Prover always gets a graph H, which is isomorphic to G_0 and G_1 independent of b. Prover cannot derive anything. Can only estimate b with probability $\frac{1}{2}$ and $\tilde{b}$ is the guessed value. Probability with error $= \frac{1}{2}$.
2. $G_0 \ncong G_1$: Prover is capable of determining bit b of the Verifier and sets $\tilde{b} = b$. Therefore Veronica accepts. Probability for an error = 0.

For $\overline{GI} \in$ IP(2) private random bits of V are essentially important.

Definition. (notation) AM[K] or AM(k).

Definition. AM(k) results from IP(k) if the verifier has no access to private random bits. Obvisouly AM(k) $\subseteq$ IP(k)

Theorem 4.3. (Goldwasser, Sipser)

$$\mathrm{IP(k)} \subseteq \mathrm{AM(k+2)}$$

In the practicals we will consider an AM(2) protocol for $\overline{\mathrm{GI}}$.

13.5 Regarding complexity of GI

The following theorem holds:

Theorem 4.4. (without proof) If GI is NP-complete, then $\Sigma_2 P = \Pi_2 P$. The proof uses an AM(2) protocol.

IP protocols can be traced back to research by Goldwasser, Micali and Radkoff. AM protocols were considered by Babai.

13.6 4.3 Interactive protocols with zero knowledge property (ZKP)

We distinguish between perfect ZKP and its attenuations. In perfect ZKP the prover doesn't want to share "any" information with the verifier (to avoid abuse). For example if the question is: Does any hamiltonian cycle exist? But prover does not want to share the hamiltonian path itself. So generally, we want to show that some X satisfies some property, but we don't want to share X itself. X might be a certificate of a NP problem.

13.7 ZKP Problem: Magic door

Given. Given is a room A and room B. They are separated by a magic door which can only be opened by a secret. An anteroom is externally accessible and allows to enter room A and room B through separate (simple) doors.
Question. How can Bob verify that he knows the secret without telling Alice the secret?

Alice stays in the anteroom. Bob goes into room A and comes back from room B. He must have changed the room through the magic door.

1. Prover and Verifier is in the external area. Prover enters the anteroom and closes door behind himself. Prover choses bit i randomly and enters room i. He closes the door behind himself.
2. Verifier enters the anteroom and choses bit j randomly and tells Prover her choice.
3. Bob appears from one of the doors.
4. Verifier accepts, if Bob comes from room j.

Claim. This protocol has perfect zero knowledge property.

Definition. (perfect zero knowledge property) An interactive protocol has the perfect ZKP (=PZKP) iff the following algorithm does not reveal any information:

1. Consider a malicious Verifier, which executes (instead of V) any other arbitrary efficient algorithm (with the intention to determine information about Bob).
2. There is an efficient simulation algorithm which produces exactly the same messages like the protocol (B, V').

An interactive protocol for a decision problem L has PZKP iff there is a randomized simulation algorithm S for every randomized algorithm V' with polynomial runtime, whose expected maximum runtime is polynomial and which for $x \in L$ can evaluate all information shared via messages between Prover and Verifier with the same probability.

Defintion. (PZK) Class of all problems with an interactive protocol for which the verification is possible in polynomial time and the protocol is a PZKP.

For our example: This protocol is a correct interactive protocol and is a perfect zero knowledge protocol.

Regarding interactive protocols:

Case 1 Prover does not posses secret and detects it with probability $\frac{1}{2}$.

Case 2 Prover knows secret. He can also come from the correct room and the Verifier will accept it.

Regarding PZK: We need a Prover simulation ("Prover Double") which looks like Prover but does not know secret.

Bob enters anteroom. Alice accepts if Bob's Double comes from door j. If Alice rejects, we start a new trial. We have a finite number of trials.

Theorem 4.5. GI $\in$ PZK.

Proof. Constructive proof by providing an appropriate protocol.

Given. graphs G_0, G_1 (n vertices).

Prover selects one permutation $\pi \in S_n$ randomly and determines $\phi_1 = \phi(G_1)$. He sends ϕ_1 to verifier. Verifier randomly choses a bit $b \in \{0, 1\}$ and sends b to the prover. If $b = 1$, prover sends ϕ_1 to verifier otherwise it sends $\pi_2 = \pi_1 \circ \pi$. π was selected (by prover) in such a way that $G_1 = \pi(G_0)$. In case that this proves isomorphism, the prover is in possession of an appropriate π.

$$\text{verifier accepts} \Leftrightarrow H = \pi_2(G_b)$$

Remark. We could also let the prover choose one random bit and let him permute the graph. However, this does not change the theoretical points of it.

If G_0 is isomorphic to G_1 the prover can make verifier to accept every input. Otherwise the probability to detect the false claim is $\frac{1}{2}$ (as always: repetition is possible).

Regarding PZK: $\tilde{V}$ is an arbitrary V algorithm. We have to define a simulation algorithm S for prover (ie. the corresponding protocol).

Simulation algorithm. Choose bit $\tilde{b} \in \{0, 1\}$ and random permutation $\tilde{\pi} \in S_n$. Determine $H = \tilde{\pi}(G_{\tilde{b}})$. ϕ_1 is input for V^*. V^* provides bit $b^* \in \{0, 1\}$. If

$\tilde{b} = b^*$, S^* sends $\tilde{\pi}$ to V^* and provides as a decision whatever V^* has provided. If $\tilde{b} \neq b^*$, discards trial. Restart and try again.

We obtain $\tilde{b} = b^*$ with probability $\frac{1}{2}$. Probability for k iterations is 2^{-k}. Expected runtime is $T(n) = \sum_{k=1}^{\infty} 2^{-k} = 2$.

Interesting is only the case if G_0 is isomorphic to G_1 because otherwise there is no secret in runtime of V^*.

Regarding distribution of messages. The first message of S^* has the same distribution like in case of the first message of the prover. The message corresponds to a random graph, which is isomorphic to G_0 and G_1. With ϕ_1 we don't share $\tilde{b}$.

$$\tilde{b} = b^* \text{ with probability } = \frac{1}{2}$$

In this case ($\tilde{b} = b^*$) the messages ϕ_1 and $\tilde{\pi}$ which gets received by V^* is identical distributed to messages, which result from the real communication between prover and verifier.

Remark. Further examples for PZK protocols: see practicals.

Is there any PZK protocol for HAMILTONIAN PATH? We guess not (as for any other NP-complete problem).

$$\underbrace{\mathrm{GI} \in \mathrm{IP}(1)}_{\text{because in NP}} \cap \underbrace{\mathrm{co-IP}(2)}_{\text{we have shown } \overline{\mathrm{GI}} \in \mathrm{IP}(2)}$$

$$\text{GI NP-complete} \Rightarrow \Sigma_2\mathrm{P} = \Pi_2(\mathrm{P})$$

If there is a ZPK protocol for NP-complete problems, polynomial hierarchy would collapse to layer 2. Is some less strict definition satisfied for NP-complete problems?

First idea for attenuation. Leads to SZK(P) (statistical zero knowledge property). Simulator's generated distribution of messages has insignificant distance to distribution of the real protocol.

$\varepsilon(n)$ is insignificant if $\varepsilon(n)$ is superpolynomially smaller. Therefore for all polynomials p and a sufficiently large n:

$$\varepsilon(n) < \frac{1}{p(n)}$$

The complexity SZK has some theoretically interesting properties. For example we assume that this class lies exactly between P and NP. But it is (like for PZK) unlikely that there exists protocols in SZK for NP-complete problems.

$$\mathrm{SZK} \subseteq \mathrm{IP}(2) \cap \mathrm{co}-\mathrm{IP}(2)$$

Second idea of attenuation. Leads to CZK(P) (computational zero knowledge property) Randomized algorithm with polynomial runtime can only distinguish the distribution generated by the simulator from the distribution of the real protocol with insignificant probability.

It turns out that (under the assumption mentioned below) CZK protocols exist for NP-complete problems.

Definition. (Standard assumption in formal cryptography) There exists a one-way function[7].

There are several approaches to define one-way functions formally.

- A function $f : \{0,1\}^* \to \{0,1\}^*$ (computable in polynomial time) is a one-way function, if for all randomized polynomial algorithms A we can state that

$$\text{probability}(A(y) = x' \text{ with } f(x') = y) < \varepsilon(n)$$

$$x \in \{0,1\}^* \quad y = f(x) \quad \varepsilon \text{ is very small}$$

Typical candidates for one-way functions:

- Integer factorization
- Robin functions
- Discrete logarithm

13.8 Complexity class UP

Definition. (complexity class UP) A non-deterministic turingmachine is named **u**nambiguous, iff for every input x there is exactly one accepting computational path. UP is the class of languages which can be decided by unambiguous NTMs in polynomial time.

$$\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$$

In the following consider the following variants for the concept of a one-way function f.

[7] $f(x)$ is computationally "easy" to determine, but $f^{-1}(x)$ is computationally infeasible. Easy means for example a deterministic polynomial algorithm. Infeasible means for example a randomized algorithm with polynomial runtime cannot compute x.

1. f is injective and for $x \in \{0,1\}^* : |x|^{\frac{1}{k}} \leq |f(x)| \leq |x|^k$ for some $k > 0$.
2. $f \in$ FP.
3. $f^{-1} \notin$ FP.

Theorem 4.6. UP = P $\Leftrightarrow$ there does not exist a one-way function (as defined above).

Proof. There exists a one-way function f. Consider a language L_f with

$$L_f = \{(x, y) : \exists z \text{ with } f(z) = y \wedge z \preccurlyeq x\}$$

$\preccurlyeq$ is defined here by length and (secondary) lexiographically ($0 < 1 < 00 < 01 < 10 < 11 < 000 < \ldots$).

Claim. $L_f \in$ UP $\setminus$ P.

Proof. We can easily show that $L_f \in$ UP (ie. there exists an unambiguous TM U, which accepts L_f). U guesses z for input (x, y) such that $|z| \leq |y|^k$ and tests whether $y = f(z)$. If result is yes, we test whether $z \preccurlyeq x$ and accept.

Assumption. There exists a polynomial algorithm for L_f.

Claim. We can invert f with binary-search-like approach. For given y we ask whether $(1^{|y|^k}) \in L_f$ (a string with $|y|^k$ ones). If answer is no, then we conclude that no x exists with $f(x) = y$ (if one would exist, it would satisfy $x \leq 1^{|y|^k}$) because $|y| \geq |x|^{\frac{1}{k}}$. If the answer is No, we ask

$$(1^{|y|^{k-1}}) \in L_f, \ldots$$

until one request $(1^{l-1}, y) \in L_f$ results in answer No. Length l of x with $1^{|y|^k}$. After $2n^k$ calls of a polynomial algorithm for L_f x is determined (an x with $f(x) = y$ if one exists).

So we have inverted f to y in polynomial time. $f^{-1} \in$ FP is a contradiction to f as a one-way function.

We now have to show that $\exists$L $\in$ UP $\setminus$ P. Is U an unambiguous turing machine which accepts L and x is an accepting computation of U to the input y. We define a function f_u with

$$f_u(x) = \underbrace{1}_{\text{prefix}} y$$

- Therefore we have a polynomial dependency between parameter and function value because the computations of U run in polynomial time.
- f_u is injective, because U is unambiguous and we use 0 as 1 at the beginning (prefix) as a flag. We use $0y$ for an accepting computation.

- If we can invert f_u in polynomial time, we could decide L in deterministic polynomial time, because inverting of f_u to $1y$ tells us whether U accepts y or not.

This deterministic polynomial time is a contradiction to UP $\setminus$ P.

Remark. The existence of a one-way function is also relevant for other areas. For example the existence of so-called pseudo random numbers generators and the Goldreich-Levin theorem.

13.9 Returning to CZK problems

Important aspect: bit commitment.

2 operations:

fixitation (Festlegung) Prover chooses 1 bit $b \in \{0, 1\}$. Instead of sending the bit, he sends a bitstring $c(b, K)$ to the verifier with k as secret key of prover.

revelation (Aufdeckung) At any later point in time verifier can request (in case of doubt), that the prover publishes his secret b.

Desired properties.

- Fixitation should be "hidden": Without knowledge of the key, verifier should not be able to retrieve any information from $c(b, K)$ about b. More precisely: For a uniformly distributed K a randomized algorithm with polynomial runtime has an insignificant probability to guess b from $c(b, K)$.
- Fixitation should be bound. For *no* K it results that $c(b, K) = c(1-b, K')$.

Under the assumption that an one-way function exists, there exists such a bit fixitation method (eg. using RSA).

Theorem 4.7. (Relevant for RSA-like approach) For the Hamiltonian cycle problem there exists a CZK protocol under the cryptographical standard assumption (of one-way functions).

Proof. Consider the following interactive protocol. Is $G = (V, E)$ an undirected graph with n vertices. We assume that a functional bit commitment method is provided. Now:

1. Bob chooses random permutations $\pi \in S_n$ and permutes G ($\rightarrow \pi(G)$). He sends for π and the vertices list of $\pi(G)$ the bit commitment to Verifier.

2. Veronica selects randomly a bit $i \in \{0, 1\}$ and sends it to Bob.
3. If $i = 0$, Bob publishes the bits for π and $\pi(G)$. If $i = 1$, Bob publishes the bits of n edges of $\pi(G)$ (this corresponds to the number of n edges of G). If G has an hamiltonian cycle he also publishes the edges of the hamiltonian cycle.
4. If $i = 0$, verifier accepts, if permutation $\pi' \in S_n$ and the corresponding edge list for $\pi'(G)$ was published. If $i = 1$, verifier accepts if the prover provided the edges of a hamiltonian cycle. In other cases the verifier rejects.

Correctness as interactive protocol:

1. If G has a hamiltonian cycle, the prover and verifier can always follow the protocol.
2. If there is no hamiltonian cycle, prover cannot prepare himself for $i = 0$ and $i = 1$. If prover permutes the correct graph correctly, he can pass the test for $i = 0$, but will fail for $i = 1$. A malicious prover will be detected in case $i = 0$ and will pass $i = 1$. We get an one-sided error probability $\frac{1}{2}$ (can be reduced by repetition). Therefore our protocol is correct.

Assumption: G has hamiltonian cycle. The prover uses a random secret of fixed length. Consider the protocol (B, V'). Consider the prover's revelation as the secret publication of bit fixitation for π and $\pi(G)$. Simulation algorithm:

1. Simulation of prover: Select random bit $i \in \{0, 1\}$ and work with hypothesis, such that V' algorithm will select $i = i'$. If $i' = 0$ for random $\pi \in S_n$, send bit fixitation for $(\pi, \pi(G))$. If $i' = 1$ for random $\pi \in S_n$, send bit fixitation for $(\pi, \pi(H'))$ with H' as the graph only consists of the hamiltonian cycle $1, 2, \ldots, n$.
2. Simulate V' for the simulated transmitted data by prover.
3. If $i \neq i'$, we make a new start with step 1.
4. Simulate algorithm of prover. Uncover the information (as given by i).

Following from the properties of bit commitment, probability$(i = i') = \frac{1}{2}$ is the expected number of iterations constant. Distribution of messages of simulation algorithm S vs protocol (B, V'). The difference is only with insignificant probability distinguishable for V. Analogously we can define CZK protocols for all other NP-complete problems.

As a second example: Colorization of a graph with 3 colors.
Given. Undirected graph $G = (V, E)$
Find. 3-colorization of G: ϕ (with $V' \to \{1, 2, 3\}$) such that

$$\{i, j\} \in E \Rightarrow \phi(i) \neq \phi(j)$$

Consider the following interactive protocol

1. The prover selects a random permutation π of $\{1, 2, 3\}$. For $i = 1$ to n the prover sends bit fixitation for $\pi(\phi(i))$ to the verifier. [If Bob has 3-colorization, he takes those as ϕ].
2. Verifier select random edge $e \in E$ and sends it to the prover.
3. $e = \{i, j\}$. Prover has to publish the values transmitted in step 1 for i or j.
4. Verifier check whether the uncovered colors are different and equate with the values of step 1.
5. If yes, verifier accepts, else rejects them.

Probability for malicious prover detection is very small. If we repeat $t = |E|$ times, then error probability $\sim e^{-t}$.

Implementation of this stuff follows in RSA-like approach.

Possible approaches:

Factorization Prover chooses randomly a large prime number p. p is in binary representation

$$p = (\underbrace{p_{l-1}, \ldots, p_0}_{\text{digits}})$$

p is large enough such that $b = p_0 \oplus p_1 \oplus \ldots \oplus p_{l-1}$. Furthermore prover selects prime number q with $q < p$. Determine $n = pq$. Fixitation: Prover sends n. Revelation: Bob sends p, q. (Is hidden under the assumption that factorization of n is difficult and bound because factorization is distinct)

RSA-like 3 colors $= \{00, 11, 01\}$. Prover generates random permutation π of the color set and n pairs of edges of RSA key pairs (p_i, q_i, d_i, e_i) for $i \in V$. For each vertex $i \in V$, prover computes encryption (y_i, y_i') of the color of i under π (when using the corresponding RSA-system). Consider $b_i b_i'$ as the two bits of $\pi(\phi(i))$, then $y_i = (2x_i + b_i)^{e_i} \mod p_i q_i$ and $y_i' = (2x_i' + b_i')^{e_i} \mod p_i q_i$ with x_i, x_i' random integers $\leq \frac{p_i q_i}{2}$ (private computation of the prover). Prover publishes $(e_i, p_i, q_i, y_i, y_i')$ for $i \in V$ (ie. public RSA key and encrypted colors). The verifier randomly selects edge $\{i, j\} \in E$. Prover now provides verifier d_i and d_j. Verifier can now compute:

$$b_i = (y_i^{d_i} \mod p_i q_i) \mod 2$$

$$b_i' = (y_i'^{d_i} \mod p_i q_i) \mod 2$$

Analogously b_j, b_j'

She tests whether $b_i b_i' \neq b_j b_j'$. Verifier does not learn anything about the colorization of the prover.

Conclusio. If we allow 2 or more independent proofs, we come up with multi-prover interactive systems (complexity class mIP). This provides us NEXP (very powerful!).

14 PCPs and the PCP theorem

Definition. (PCP) probabilistically checkable proofs. Or: randomized verifiable proofs.

Definition. Is $r, q : \mathbb{N} \to \mathbb{N}$. A $(r(n), q(n))$-bounded PCP is a randomized verifiable algorithm V with polynomial runtime and the following properties:

- For the input x of length n and a proof $B = \{0,1\}^k$, the algorithm V has access to x and a random vector $r \in \{0,1\}^{r(n)}$ (ie. V has $r(n)$ random bits available).
- Based on this information V computes up to $\mathcal{O}(q(n))$ positions and gets the corresponding bits of the proof as additional information (in general V does not know the whole proof).
- Finally V computes the decision whether x gets accepted.

$$V(x, r, B) \in \left\{ \underbrace{0}_{\text{reject}}, \underbrace{1}_{\text{accept}} \right\}$$

Because V must have polynomial runtime only polynomial functions are considered for $r(n)$ and $q(n)$ (this is some kind of ressource constraint).

Definition. A decision problem L is part of the class PCP($r(n)$, $q(n)$) if there exists a $(r(n), q(n))$-bounded PCP verifier V such that

$$\forall x \in L \exists \text{proof B with } \text{probability}_z(V(x, z, B) = 1) = 1$$

$$\forall x \notin L \land \forall \text{proofs B'} : \text{probability}_z(V(x, z, B')) \leq \frac{1}{2}$$

The second definition defines a constrained one-sided error.

Relation of PCP, NP and P.

$$\mathrm{NP} = \mathrm{PCP}(0, \mathrm{poly(n)})$$

$$\mathrm{P} = \mathrm{PCP}(0, \log\mathrm{(n)})$$

PCP theorem. $\mathrm{NP} = \mathrm{PCP}(\log \mathrm{n}, 1)$.

We are going to prove $\mathrm{NP} = \mathrm{PCP}(\mathrm{n}^3, 1)$.

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

Proof. Is $p(n)$ the runtime of $V \to p(n)$ as polynomial in n. In total $\leq g(n) \cdot 2^{\mathcal{O}(r(n))}$ are read.

Guess those bits non-deterministically (random bitstring). Simulate for ever random bitstring the control flow of V and accept if all simulations return acceptance. Computational runtime boundary: $2^{\mathcal{O}(r(n)+\log n)}$.

Consequence.

$$\mathrm{NP} = \mathrm{PCP(\log n, poly(n))} = \bigcup_{\mathrm{k} \geq 0} \mathrm{PCP(\log n, n^k)}$$

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

We construct a $\mathrm{PCP(n^3, 1)}$ verifier for 3SAT. A 3SAT instance is given.

$$C_1, C_2, \dots, C_m \text{ clauses}$$
$$x_1, x_2, \dots, x_n \text{ variables}$$

Idea. Arithmetization of 3SAT equation.

boolean	arithmetic
x_i	$1 - x_i$
$\overline{x_i}$	x_i
$\wedge$	$+$
$\vee$	$\cdot$

Arithmetic returns polynomial of maximum degree 3.

Observation. Is $a \in \{0,1\}^n$. $P(a) = 0 \Leftrightarrow$ a is a satisfying assignment for ϕ. $P(a) > 0$. $P(a)$ returns the number of satisfying clauses.

Transition to arithmetic modulo 2.

$$P(a) \equiv 1 \mod 2 \qquad \text{a is not satisfying (no error possible)}$$
$$P(a) \equiv 0 \mod 2 \qquad \text{not all clauses must be satisfied}$$

Idea. We introduce randomization. We hide randomly a set of clauses.

Let P_i be the polynomial corresponding to the ith clause (after arithmetization). ρ is a random vector $\in \{0,1\}^m$ (for m clauses). $P^{(\rho)}$ is the sum of all polynomials P_i with $\rho_i = 1$ (terms of all visible clauses).

$$P^{(\rho)} = \sum_{i=0}^{m} \rho_i \cdot P_i$$

For satisfying assignments a, $P^{(\rho)}(a) = 0$. probability($P^{(\rho)}(a) \equiv 0 \mod 2$) = probability($P^{(\rho)}(a) \equiv 1 \mod 2$) = $\frac{1}{2}$.

We still have the problem that we depend on a and a requires n bits and is therefore inappropriate for our purposes.

We need method to compute $p(a)$ (=$p^{(\rho)}(a)$) without knowledge of a.

V must be capable to derive from the available information a constant (here 3) number of proof positions and from the values of those proof positions the decision about acceptance.

In general: Consider a polynomial $\Psi = (x_1, x_2, \ldots, x_n)$ of degree ≤ 3 over $\mathbb{Z}_2$ (p and $p^{(\rho)} \mod 2$ of this structure). Ψ consists (probably multiplication is necessary) of the following terms:

$$x_i \in \{0,1\} \text{ constant term}$$

$$x_i i \in I_\Psi^1$$
$$x_i \cdot x_j (i,j) \in I_\Psi^2$$
$$x_i \cdot x_j \cdot x_k (i,j,k) \in I_\Psi^3$$

Define now linear functions L_1^a, L_2^a, L_3^a.

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

$$L_2^a : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2 \qquad L_2^a(y_{11}, \ldots, y_{nn}) = \sum_{i=1}^n \sum_{j=1}^n a_i \cdot a_j \cdot y_{ij}$$

$$L_3^a : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2 \qquad L_3^a(y_{111}, \ldots, y_{nnn}) = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n a_i \cdot a_j \cdot a_k \cdot y_{ijk}$$

We can store those function as function tables. For L_1^a length 2^n, for L_2^a length 2^{n^2} and for L_3^a length 2^{n^3}. So the total length is $2^n + 2^{n^2} + 2^{n^3}$. $\gamma_\Psi^1, \gamma_\Psi^2$ and γ_Ψ^3 are the characteristical vectors of I_Ψ^1, I_Ψ^2 and I_Ψ^3 (for example γ_Ψ^2 has a One at position $(i,j) \Leftrightarrow (i,j) \in I_\Psi^2$). Now we can state $\Psi(a) = \gamma_\Psi + L_1^a(\gamma_\Psi^1) + L_2^a(\gamma_p si^2) + L_3^a(\gamma_\Psi^3)$. $\Psi(a)$ can be evaluated without access to a. V can evaluate $L_1^a(\gamma_\Psi^1), L_2^a(\gamma_\Psi^2)$ and $L_3^a(\gamma_\Psi^3)$ (which is $p(a)$ or $p^\gamma(a)$) without using a.

We have a function table for L_1^a, L_2^a and L_3^a. We check: $p^{(\rho)}(a)$ for constant, randomized ρ and accept only if all values are zero.

The number of required random bits is $\mathcal{O}(n^3)$, because number of clauses in non-trivial 3SAT instance is

$$\leq 2n + 4\binom{n}{2} + 8\binom{n}{3} = \mathcal{O}(n^3)$$

We need an approach for proofs of random kind. We can assume that proof have the "right" length.

A proof verifier for the general case consists of the following components:

- linearity testing
- robust function evaluator
- consistency test
- proof verifier for well-formed proofs

14.1 Linearity testing

Definition. $f : \mathbb{Z}_2^m \to \mathbb{Z}_2$ is linear if $f(x) + f(y) = f(x + y) \forall x, y \in \mathbb{Z}_2^m$. f is δ-close to a function g ($f, g : \mathbb{Z}_2^m \to \mathbb{Z}_2$) if (for a uniform distribution of x to $\mathbb{Z}_2^m$) we can state:

$$\text{probability}_x(f(x \neq g(x))) \leq \delta$$

Followingly we call a function almost-linear, if for a usefully selected δ (we will discuss this later on) the function is δ-close to a linear function.

Idea of linearity tests. Select randomly and independent of each other $x, y \in \mathbb{Z}_2^m$ and name f non-linear if $f(x) + f(y) \neq f(x + y)$ (otherwise linear). In this case, the linearity test fails.

Properties:

- If f is linear, dann f will pass the test.
- If f is not δ-close for any $\delta < \frac{1}{3}$ to a linear function, then f will pass the test with probability $1 - \frac{\delta}{2}$.

To show the first property is immediate. For the second property we have to show probability$(f(x + y) \neq f(x) + f(y)) \leq \frac{\delta}{2}$ (f is δ-close to linear function g). This is a constructive proof. We define a linear function g, which is δ-close to f. $a \in \mathbb{Z}_2^m$. We define $g(a)$ as follows: Evaluate all function values $f(a + b) - f(b)$ with $b \in \mathbb{Z}_2^m$ (fyi, $f(a + b) \in [0, 1]$ and $f(b) \in \mathbb{Z}_2$). We set $g(a) = 0$ if 0 occurs more often (or equal) as result for $f(a + b) - f(b)$ then 1. Otherwise $g(a) = 1$.

Remainingly we have to show that

- g is linear to u.
- g is δ-close to f.

Regarding point 1: Assume f and g are not δ-close to each other,

$$\Rightarrow \text{probability}_x(f(x) \neq g(x)) > \delta$$

Because of the construction g, we can state that

$$\text{probability}_y(g(a) = f(a+y) - f(y)) \geq \frac{1}{2}$$

Therefore

$$\begin{aligned}
&\text{probability}_{x,y}(f(x+y) - f(y) \neq f(x)) \\
&\geq \text{probability}_{x,y}(f(x+y) - f(y) = g(x), g(x) \neq f(x)) \\
&= \sum_{a \in \mathbb{Z}_2^m} \text{probability}_y(f(a+y) - f(y) = g(a), g(a) \neq f(a))
\end{aligned}$$

For all $a \in \mathbb{Z}_2^m$ we can state:

$$f(a) = g(a) \vee f(a) \neq g(a)$$

In the first case, the probability (from above) is 0 and in the second case we can omit the precondition $f(a) \neq g(a)$.

$$\begin{aligned}
&\text{probability}_{x,y}(f(x+y) - f(y) \neq f(x)) \\
&\geq \frac{1}{2^m} \cdot \sum_{a \in \mathbb{Z}_2^m, f(a) \neq g(a)} \text{probability}_y(f(a+y) - f(y) = g(a)) \\
&\geq \frac{1}{2^m} \cdot \sum_{a \in \mathbb{Z}_2^m, g(a) \neq f(a)} \frac{1}{2} > \frac{\delta}{2}
\end{aligned}$$

The last inequation follows, because from

$$\text{probability}_x(f(x) \neq g(x)) > \delta$$

it follows that more than $2^m \cdot \delta$ of all $a \in \mathbb{Z}_2^m$ it requires $f(a) \neq g(a)$. We get a contradiction to $\text{probability}_{x,y}(f(x+y) \neq f(x) + f(y)) \leq \frac{\delta}{2}$.

For the second bullet point: To determine the linearity of g we consider

$$p(a) = \text{probability}_x(g(a) = f(a+x) - f(x))$$

Obviously, $p(a) \geq \frac{1}{2}$ (choice of g). We want to show a stronger statement: $p(a) \geq 1 - \delta$.

$$\underbrace{\text{random } x \in \mathbb{Z}_2^m}_{\text{uniformly distri.}} \rightarrow \underbrace{x + a \in \mathbb{Z}_2^m \text{ is random}}_{\text{uniformly distri.}}$$

$$\text{probability}_{x,y}(f(x+a)+f(y)+f(x+a+y)) \leq \frac{\delta}{2}$$

We call it event 1. This is a requirement in the inequation of bullet point 2.

Analogously, event 2,

$$\text{probability}_{x,y}(f(x)+f(y+a) \neq f(x+a+y)) \leq \frac{\delta}{2}$$

The probability for the union of both events is bounded by above δ and the probability of the complement is bounded below by $1-\delta$.

According to the DeMorgan rules, the intersection of $f(x)+f(y+a)$ is $f(x+a+y)$ and $f(x+a)+f(y)$ is $f(x+a+y)$. This corresponds to the subset of the event $f(x+a)+f(y) = f(y+a)+f(x)$.

$$\begin{gathered}
\text{probability}_{x,y}(f(x+a)+f(y) = f(y+a)+f(x)) \geq 1-\delta \\
\Leftrightarrow f(x+a)-f(x) = f(y+a)-f(y) \\
\Rightarrow P = \text{probability}_{x,y}(f(x+a)-f(x) = f(y+a)-f(y)) \geq 1-\delta \\
P = \sum_{z\in\{0,1\}} \text{probability}_{x,y}(f(x+a)-f(x) = z, f(y+a)-f(y) = z)
\end{gathered}$$

$$\overset{\text{iid}}{=} \sum_{z\in\{0,1\}} \text{probability}_x(f(x+a)-f(x)=z).\,\text{probability}_y(f(y+a)-f(y)=z)$$

$$= \sum_{z\in\{0,1\}} [\text{probability}_b(f(x+a)-f(x)=z)]^2$$

For $z = g(a)$ is $\text{probability}_x(f(x+a)-f(x)=z) = p(a)$. For $z \neq g(a)$ we get $\text{probability}_x(f(x+a)-f(x)=z) = 1-p(a)$.

Therefore it follows from the derivation so far:

$$1-\delta \leq p(a)^2 + (1-p(a))^2$$

(we know that $p(a) \geq \frac{1}{2}$.) Thus

$$1-p(a) \leq \frac{1}{2} \leq p(a)$$

$$\begin{gathered}
\Rightarrow p(a)^2 + (1-p(a))^2 \leq p(a)^2 + p(a)(1-p(a)) \\
p(a)^2 + (1-p(a))^2 = p(a)
\end{gathered}$$

So we have proven $p(a) \geq 1-\delta$. We will use this result now at three different occations:

$$p(a) = \text{probability}_x(g(a) = f(a+x)-f(x)) \geq 1-\delta$$

$$p(b) = \text{probability}_x(g(b) = f(b + a + x) - f(a + x)) \geq 1 - \delta$$
$$p(a + b) = \text{probability}_x(g(a + x) = f(a + b + x) - f(x)) \geq 1 - \delta$$

The intersection of all these three events has probability $\geq 1 - 3\delta$. Add the first two events and substract the third event:

$$\text{probability}_x(g(a) + g(b) = g(a + b)) \geq 1 - 3\delta$$

Resulting from the requirement $\delta < \frac{1}{3}$ we get:

$$\text{probability}(g(a) + g(b) = g(a + b)) > 0$$

Because the inner condition is independent of x, we get

$$\text{probability}_x(g(a) + g(b) = g(a + b)) = 1$$

and therefore g is linear

$$g(a + b) = g(a) + g(b) \forall a, b$$

The linearity test is proven.

14.2 Robust function evaluator

Goal. For $\delta < \frac{1}{3}$ the following properties have to be satisfied:

- If f is linear, then function evaluator has to provide $f(a) \forall a \in \mathbb{Z}_2^m$.
- If f is δ-close to linear function g, then function evaluator (randomized algorithm) has to provide value $g(a)$ with a error probability bounded by 2δ.

We take the following approach to determine $f(a)$: Select $x \in \mathbb{Z}_2^m$ randomly and evaluate $f(x + a) - f(x)$. It remains to show the two cases we just stated in the bullet points.

The first case is trivial, because f is linear: $f(x + a) = f(x) + f(a) \forall a$.

For the second case f and g are δ-close:

$$\text{probability}_x(f(x) = g(x)) \geq 1 - \delta$$

$$\text{probability}_x(f(x + a) = g(x + a)) \geq 1 - \delta$$

The probability that both events will occur is $\geq 1 - 2\delta$. From those two events it follows

$$f(x+a) - f(x) = \underbrace{g(x+a) - g(x)}_{=g(a)}$$

Because g is linear: $f(x+a) - f(x) = g(a)$. Function evaluator has been proven.

14.3 Consistency test

We have 3 function stabilizors f_1, f_2 and f_3 (in our proof this is L_1^a, L_2^a, L_3^a). f_1, f_2 and f_3 are each linear or δ-close for a linear function (otherwise they will fail the linearity test).

For the consistency test we assume $\delta < \frac{1}{24}$.

1. If function stabilizors of f_1, f_2 and f_3 are linear functions of type L_1^a, L_2^a and L_3^a (for any $a \in \{0,1\}^n$) the consistency test has to succeed.
2. If there is no $a \in \{0,1\}^n$ such that the functions (represented by the function tables of f_1, f_2, f_3) are δ-close to L_1^a, L_2^a and L_3^a, the consistency test has to succeed with error probability bounded by a constant c.

Construction. We choose randomly and independent $x, x', x'' \in \mathbb{Z}_2^m$ and $y \in \mathbb{Z}_2^{n^2}$.

Define $x = x'$ by

$$(x = x')_{ij} = x_i \cdot x'_j \to \dim n^2$$

$$x'' \circ y \text{ by } (x'' \circ y)_{ijk} = x''_{ijk}$$

Use function evaluator to get estimator for

$$b \Rightarrow f_1(x) \qquad b' \Rightarrow f_1(x') \qquad b'' \Rightarrow f_1(x'') \qquad c \Rightarrow f_2(x \circ x')$$

$$c' \Rightarrow f_2(y) \qquad d \Rightarrow f_3(x'' \circ y)$$

The consistency test will succeed if $bb' = c$ and $b''c' = d$.

Remark. The linear functions L_1^a, L_2^a and L_3^a will succeed the test of course. For them the function evaluation value is error free and we can state:

$$L_1^a(x) \cdot L_2^a(x') = \left(\sum_{i=1}^{n} a_i x_i\right) \cdot \left(\sum_{j=1}^{n} a_j x'_j\right) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j v_i x_j$$

Analogously

$$L_1^a(x'') \cdot L_2^a(y) = L_3^a(x'' \circ y)$$

For our two desired properties for the consistency test: Because $\delta < \frac{1}{24}$, $2\delta < \frac{1}{12}$. Because we have 6 calls of the function evaluator and each of them has error probability $< 2\delta < \frac{1}{12}$, the total error probability for function evaluations is $\frac{1}{2}$.

Consider now the case that all function evaluations are passed successfully. A linear δ-close function f_1 has coefficient a_i. A δ-close function f_2 has coefficient b_{ij} (Matrix $B = (b_{ij})$).

Define a matrix $A = (a_{ij})$ with $a_{ij} = a_i \cdot a_j$.

Remark. Function evaluations is correct, but function stabilizor is inconsistent. We now consider the case $bb' \neq c$. Then follows $A \neq B$. Consider x and x' as column vectors, then the consistency test compares $x^t A x'$ and $x^t B x'$. This test is based on the fact that $A \neq B$ and random x and x' with probability $\geq \frac{1}{4}$ the values are difference and therefore inconsistency is shown.

If A and B differe in the j-th columns, the probability that $x^t A$ and $x^t B$ differ at the j-th position is $\frac{1}{2}$.

Combining all these thoughts, we get a $\mathrm{PCP}(\mathrm{n}^3, 1)$ verifier for 3SAT.

Remark. $\mathrm{PCP}(\mathrm{poly(n)}, 1) = \mathrm{NEXP} = \bigcup_{\mathrm{c} \geq 1} \mathrm{PCP}(\mathrm{n}^{\mathrm{c}}, 1)$.

In the approximation chapter we will have a fresh look at the PCP theorem from a different perspective (reformulation with focus on non-approximative results).

15 Approximation from complexity theory perspective

Motivation. Many optimization problems are different to solve exactly.

Similar Question. Can we efficiently compute approximative solutions for such problems?

Basic question. How do we measure the quality of an approximation? Most basically this is the relative error.

Instance 1	Opt. value 17	Approx. solution: 27
Instance 2	Opt. value 1700	Approx. solution: 1710

Table 2: Example for quality of an approximation. Instance 2 has the smaller relative error.

In the following we will discuss optimization problems. x is the input. $S(x)$ denotes the set of valid solutions for the instance with input x.

$$S(x) \neq 0 \qquad \text{Not optimizable in the other case}$$

Denote the target function value of the solution with $s \in S(x)$. $V_{\text{opt}}(x)$ denotes the optimal target function value for the input x. Furthermore we assume $V(x, s) > 0 \forall x, s$ (not required, but makes quality function less complex).

$\forall x, s \in S(x)$ the encoding length of s and $v(x, s)$ is polynomial in $|x|$ (Background: our approximation algorithm shall have polynomial runtime).

Definition. Approximation quality (here: one of multiple variations)

$$r(x, s) = \frac{v(x, s)}{v_{\text{opt}}(x)} \text{ with } s \in S(x) \text{ for min opt problems}$$

$$r(x, s) = \frac{v_{\text{opt}}(x)}{v(x, s)} \text{ with } s \in S(x) \text{ for max opt problems}$$

Remark.

- This definition is possible due to $v(x, s) > 0$ (no absolute value or division required).
- We always have $r(x, s) \geq 1$. The closer $r(x, s)$ is to 1, the better is s.
- $r(x, s) \leq c$ with $c \geq 1$ is "c-Approximation".

Algorithmical view. Consider optimization problem Q. x is the input of Q and A ist an algorithm for Q: A computes valid solutions $S_A(x) \in S(x)$. The approximation quality of A for the input x is given by $\hat{r}_A(x) := r(x, s_A(x))$.

Worst case consideration. For the algorithm A $r_A(n)$ is called maximum approximation quality of A.

$$r_A(n) = \max \{\hat{r_A}(x) \, || x| \leq n\}$$

Naming. If $r_A(n) \leq 1 + \varepsilon$ is given for some $\varepsilon \geq 0$ and all $n \in \mathbb{N}$, A computes an ε-optimal solution.

If $r_A(n) \leq c$ for $c \geq 1$ and all $n \in \mathbb{N}$, A computes a c-approximation (approximates Q to factor c).

Remark. Also the quality of a maximum optimization problem is defined in a way such that the quality is ≤ 1 (inverse of our definition).

Example (Bin packing). n numbers $a_i \in [0, 1]$ (item size). Bins have capacity 1.

Goal. Pack items in minimum number of bins. For the bit fit decreasing (BFD) algorithm the following statements are known:

$$v(x, s_{\text{BFD}}(x)) \leq \frac{11}{9} \cdot v_{\text{opt}}(x) + 4$$

$$r_{\text{BFD}}(x) \leq \frac{11}{9} + \frac{4}{v_{\text{opt}}(x)}$$

Without loss of generality, we assume that $v_{\text{opt}}(x) \geq 2$ because $v_{\text{opt}}(x) = 1$ instances are trivially recognizable. Then

$$r_{\text{BFD}}(x) \leq \frac{29}{9}$$

For large values of $v_{\text{opt}}(x)$ $\frac{29}{9}$ is far aways from $\frac{11}{9}$ and real quality.

We therefore introduce the asymptotic approximation quality

$$r_A^{\infty} = \inf \{b | \forall \varepsilon > 0 \exists v(\varepsilon) > 0 \forall x v_{\text{opt}} \geq v(\varepsilon), r_A(x) \leq b + \varepsilon\}$$

For example: $r_{\text{BFD}}^{\infty} = \frac{11}{9}$. When n goes to infinity, $v_{\text{opt}}(x) = \infty$ and $\frac{y}{v_{\text{opt}}(x)}$ to 0.

We will almost only consider classical quality. But there are some problems (bin packing is one of them) for which asymptotical quality is more powerful.

Definition (complexity APX, APX*). Given $r(n) : \mathbb{N} \mapsto [0, \infty]$ with $r(n+1) \geq r(n) \forall n$. The complexity class APX($r(n)$) contains all optimization problems for which an algorithm A with maximum approximation quality $r_A(n) \leq r(n)$ can be found with polynomial runtime.

$$\text{APX} := \bigcup_{\text{c>1,c const}} \text{APX(c)}$$

"In constant quality approximable in polynomial time".

$$\text{APX}^* := \bigcap_{\text{c>1,c const}} \text{APX(c)}$$

Problem for every $c > 1$ with quality c is approximable. Algorithm can depend on c.

Definition (PTAS). polynomial time approximation scheme.

A PTAS for an optimization problem Q is an algorithm (a class of algorithms) with inputs of structure (x, ε) (x is input for $Q, \varepsilon > 0, \varepsilon \in Q$) which computes a solution for Q with quality $(1 + \varepsilon)$ in polynomial time in $|x|$ ("ε optimization").

A PTAs as complexity class: contains all optimization problems for which there exists a PTAs.

Definition (FPTAS). fully polynomial time approximation scheme (runtime is polynomial in $|x|$ and in $\frac{1}{\varepsilon}$).

For PTAS runtime $\mathcal{O}(n^{\frac{1}{\varepsilon}})$ with $n = \mathcal{O}(|x|)$ is allowed. For FPTAS it is not.

Obviously $\text{P} \subseteq \text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX}$.

Example. (Max clique problem) Given an undirected graph $G = (V, E)$ with $|V| = n$. Find a complete subgraph of G with maximum vertex number.

Trivial algorithm 1. Return random vertices. Quality $\leq n$.

Trivial algorithm 2. Fixate $k \in \mathbb{N}, k < n$. Consider all subsets of V with size k and return the greatest complete subgraph $\Rightarrow$ quality $\frac{n}{k}$.

Runtime is fixed for k.

Later on we will see: $\text{MAX} - \text{CLIQUE} \notin \text{AFX}$.

Example. (Vertex cover problem) Given is an undirected graph $G = (V, E)$ with $|V| = n$. Find $V' \subseteq V$ such that every edge e in E of V' is covered (each edge in E has at least one end vertex in V' and V' has minimal cardinality).

For general graph this problem is NP-hard. The approximation algorithm is interesting.

Approximation algorithm.

1. $E' = \{\}$.
2. While there is an edge $\{u, v\}$ such that no u and v is covered by an edge in E', select this edge and add it to E'.
3. V' is set of all end vertices of edges in E'.

Claim. The algorithm above returns vertex cover V' and has approximation quality 2.

Central point is the property of E' on termination.

- E' contains all edges which that the degree of each vertex is ≤ 1. Therefore E' is maximal matching. Therefore V' is vertex cover.
- Furthermore $|E'| = k \Rightarrow |V'| \leq 2k$. Quality 2 follows, because for a cover of E' at least k vertices are necessary.

Example 3. (MAX-3SAT) Given a 3SAT equation l with variables $x_1, \dots, x_m$ and clauses $c_1, \dots, c_n$. Find an assignment such that the maximum number of clauses is satisfied.

Consider the following randomized algorithm: Assign $x_1, \ldots, x_n$ independently uniformly distributed with 0, 1. Random variable:

$$x_j = \begin{cases} 1 & \text{clause } c_j \text{ is satisfied} \\ 0 & \text{else} \end{cases}$$

Denote

$$E(x_j) = \frac{7}{8} \quad \text{7 of 8 clauses satisfied}$$

Random variable (we want X at maximum)

$$X := \sum_{j=1}^{m} x_j$$

$$E(X) = \sum_{j=1}^{m} E(X_j) = \frac{7}{8}m$$

Randomized approximation algorithm with quality $\frac{7}{8}$. We desire a deterministic algorithm.

Idea. Fixate the values x_i one after another. We have to ensure that for random selection of an assignment for the remaining variables the number of satisfied clauses has to be keep $\geq \frac{7}{8}m$.

Situation during the running algorithm: 0-3 literals per clause are fixed.

Consider $x_n = b \in \{0, 1\}$.

$$a_j = E(x_j | x_n = b) \qquad j = 1, \ldots, m$$

If with assignment $x_n = b$ the clause j is satisfied, $a_j = 1$. Otherwise there are $k = 1, 2, 3$ not-fixed literals in c_j.

Then this implies: $a_j = \frac{2^k - 1}{2^k} = 1 - \frac{1}{2^k}$. If the literal is fixed and false, $a_j = 0$.

$$E(X | x_n = b) = \sum_{j=1}^{m} E(X_j | x_n = b) = a_1 + \ldots + a_m$$

$$\text{Decidable in } \mathcal{O}(m)$$

$$E(X) = \frac{1}{2} \cdot E(X | X_n = 1) + \frac{1}{2} \cdot E(X | x_n = 0)$$

$$\text{We know: } E(X) \geq \frac{7}{8}m$$

$$\Rightarrow E(X | x_n = 1) \geq \frac{7}{8}m \quad E(X | x_n = 0) \geq \frac{7}{8}m$$

We are allowed to fixate $x_n = b_n$ such that $E(x|x_n = b) \geq \frac{7}{8}m$. So x_n is fixed. Now iterate. We gain assignment $b_1, \dots, b_n$.

$$E(x|x_1, \dots, x_n = b) \geq \frac{7}{8}m$$

Deterministic approximation algorithm with runtime $\mathcal{O}(n \cdot m)$ with quality $\frac{8}{7}$.

15.1 Example 4 Scheduling

A simple 2-machine scheduling problem as example for a problem with a PTAS.

Given. A set $J = \{1, \dots, n\}$ of tasks and we have to identical machines. A task j takes P_j time.

Goal. Find assignment of tasks such total runtime is minimal. A task cannot be split.

Problem is NP-hard (proof via partition problem).

Set $T = \sum_{j=1}^{n} p_j$. Obviously $v_{\text{opt}}(x) \geq \frac{T}{2}$.

$$\min \max \left\{ \sum_{j \in \mathbb{M}_{\mathbb{1}}} p_j, \sum_{j \in \{1,\dots,n\} \setminus M_1} p_j \right\}$$

Idea. Split tasks in 2 sets. We denote task j to be “large”, if $p_j \geq \varepsilon \cdot T$ (goal: quality $1 + \varepsilon$). Other jobs are called “small”.

Observation. There are $\leq \lfloor \frac{1}{\varepsilon} \rfloor$ large tasks.

For each of the maximal $2^{\lfloor \frac{1}{\varepsilon} \rfloor}$ possible assignments of the large tasks to the two machines, apply the following algorithm:

- Extend the currently existing machine assignment with assignment of small tasks with “least loaded” heuristics (traverse from the smallest to the greatest task and assign it to the machine with the smaller total time).
- In the end select the best solution.

Runtime is $\mathcal{O}(n 2^{\frac{1}{\varepsilon}})$ polynomial in n. So it satisfies the condition to be in PTAS.

We have to show that this is a $(1 + \varepsilon)$ approximation. Show that:

$$V(x, \underbrace{\quad s \quad}_{\text{schedule, solution}}) \leq (1 + \varepsilon) v_{\text{opt}}(x)$$

Proof of quality. Consider optimal solution:

- Corresponding assignment of large tasks to M_1 and M_2.
- Also this partition has been considered in the approximation algorithm (because *all* of them have been considered)

Case distinction:

1. All small tasks are assigned to one machine.
2. Not all small tasks are assigned to one machine.
   - The maximum difference of both assignment total time of the machines $\leq \varepsilon T$.
   - The maximum difference of completeness time of $\frac{T}{2}$ is limited by $\varepsilon \frac{T}{2}$.

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

Disadvantage: Runtime (PTAS which is no FPTAS, because of $2^{\frac{1}{\varepsilon}}$).

Example 5 Knapsack problem The knapsack problem is an example for a FPTAS. NP-complete problem.

Given are n items with weights. $g_1, \dots, g_n \in \mathbb{N}$ and values $v_1, \dots, v_n \in \mathbb{N}$. Also given is a capacity $b \in \mathbb{N}$. Find $I \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I} g_i \leq b$ and $\sum_{i \in I} v_i$ is maximal.

Classical dynamic programming algorithm:

$$F(k,y) := \max \left\{ \sum_{j=1}^{k} v_j x_j \text{ subject to } \sum_{i=1}^{k} g_i x_i \leq y, x_i \in \{0,1\} \text{ with } i = 1, \dots, k \right\}$$

Maximum value when considering the first k items. Knapsack capacity

$$F(k+1,y) = \max \begin{cases} F(k, y - g_{k+1}) + v_k + 1 \\ F(k,y) \end{cases}$$

$F(n,b)$ is decidable in $\mathcal{O}(nb)$ (pseudopolynomial) time.

Dual point of view. (to two variables k and v) $M[k,v]$ is minimal knapsack weight to reach total value.

$$M[k,v] = \min \left\{ \sum_{j=1}^{k} g_j x_j | \sum_{j=1}^{k} v_j x_j = v \right\}$$

Initialization:

$$M[0,v] = \infty \qquad \forall v > 0$$
$$M[i,v] = \infty \qquad \forall v < 0, iin\,\{1,\dots,n\}$$

Recursion:

$$M[k+1,v] = \min\{M[k,v], M[K, v - v_{k+1}] + g_{k+1}\}$$

Runtime:

$$\underbrace{\mathcal{O}(n \cdot V_{\max})}_{\underbrace{\mathcal{O}(n \cdot V_{\max})}_{\max v_j \text{ with } j\in\{1,\dots,n\}}} \quad = \mathcal{O}(n^2 \cdot v_{\max})$$

Runtime is practical iff $v_{\max}$ is not too large!

Idea. Scaling of values using t as factor for values.

t will be selected as
$$t = \frac{\varepsilon \cdot \max_{i=1,\dots,n} V_i}{(1+\varepsilon)n}$$

New resulting knapsack instance: Values $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n$, weights $g_1, \dots, g_n$ and capacity b.

Observation. $I \subseteq \{1,\dots,n\}$ is valid solution for original instance. I is the set of selected items $\Leftrightarrow$ I is a valid solution for the new problem instance (g_i, b did not get modified)

Runtime. $\mathcal{O}(n^2 \cdot \max_{i=1,\dots,n} \tilde{v}_i) = \mathcal{O}(n^2 \cdot \max\left\{\lfloor\frac{(1+\varepsilon)\cdot n \cdot v_i}{\varepsilon \cdot \max v_i}\rfloor\right\}) = \mathcal{O}(n^2 \cdot \lfloor\frac{(1+\varepsilon)n}{\varepsilon}\rfloor) = \mathcal{O}(\frac{n^3}{\varepsilon} + n^3)$ so polynomial in n *and* $\frac{1}{\varepsilon}$!. This is fine for FPTAS.

Still to show: $(1+\varepsilon)$ approximation.

Regarding quality. Is $I^* \subseteq \{1,\dots,n\}$ and optimal solution for the original instance. Is I_t the solution determined by the approximative algorithm. I^* and I_t are both valid solution for original and new instance. We can state that

$$\sum_{i\in I_t} \tilde{v}_i \geq \sum_{i\in I^*} \tilde{v}_i \text{ because } I_t \text{ is optimal solution for new instance}$$

Because $\tilde{v}_i = \lfloor\frac{v_i}{t}\rfloor$, we conclude $\frac{v_i}{t} - 1 < \tilde{v}_i \leq \frac{v_i}{t}$. Is $\text{OPT} := \sum_{i\in I^*} v_i$ value of an optimal solution.

$$\sum \tilde{v}_i \geq t \cdot \sum_{i\in I^*}(\frac{v_i}{t} - 1) = \sum_{i\in I^*} v_i - t \cdot |I^*| = \text{OPT} - t \cdot \underbrace{|I^*|}_{\leq n}$$

$$\geq \text{OPT} - \frac{\varepsilon}{1+\varepsilon} \cdot \text{OPT} = \text{OPT} \cdot \frac{1}{1+\varepsilon}$$

So we get

$$\frac{\text{OPT}}{\sum_{i \in I_t} v_i} \leq 1 + \varepsilon$$

$$\sum_{i \in I_t} v_i \geq \frac{\text{OPT}}{1+\varepsilon}$$

16 Positive and negative approximation results

Date. After christmas 2013.

In the field of approximation algorithms we distinguish "positive results" (by providing an appropriate approximation algorithm) and "negative results" (results of the structure "Problem Q cannot be approximated with quality guarantee $< c$ if … (eg. P $\neq$ NP)").

Several technique to derive negative results (non-approximations) do exist:

16.1 Gap technique

This technique is derived from the NP-hardness results. We use a gap-problem as basis.
More precisely: Given an optimization problem Q where all inputs x satisfy $v_{\text{opt}}(x) \leq a$ or $v_{\text{opt}}(x) \geq b > a$ and which is NP-complete. The problem of determining which case (a or b) is given, is called (a,b)-gap-problem.

Assuming there exists an approximation algorithm A (therefore A has polynomial runtime) with performance guarantee $< \frac{b}{a}$ for the (a,b)-gap-problem, $P = NP$ would follow. Such an approximation algorithm allows us to distinguish between both cases.

Theorem. Is A an (a,b)-gap-problem. There is no approximation algorithm for Q with performance guarantee $\frac{b}{a}$ if P $\neq$ NP.

16.1.1 Example: TSP

The Hamiltonian cycle problem is NP-complete. Given an undirected graph $G = (V, E)$ and $|V| = n$. Construct a distance matrix $n \times n$ for TSP $D = (d_{ij})$.

$$d_{ij} = \begin{cases} 1 & \{i,j\} \in E \\ n \cdot 2^n & \{i,j\} \notin E \end{cases}$$

Observation. If G is hamiltonian, the shortest tour in regards of D has length $n = 1+1+\ldots+1$. If G is not hamiltonian, every tour has length $\geq (n-1)\cdot 1+n\cdot 2^n$.

This gives us a $(n, (n-1)+n\cdot 2^n)$-gap-problem. If P $\neq$ NP there does not exist an approximation algorithm for TSP with performance guarantee 2^n.

Thus if P $\neq$ NP, $TSP \notin APX$.

Remark. For particular special cases of TSP better approximation results are possible. If D is symmetrical and triangle inequation is satisfied (metrical case), a $\frac{3}{2}$-approximation algorithm can be provided (Christofides heuristic).

Theorem. (by Arora) If d_{ij} are euclidean distances of n points, a PTAS can be provided.

16.1.2 Special case $b = a + 1$: $(k, k+1)$-gap-problem

$$b = a+1, a = k, k \in \mathbb{N}$$

Theorem. If

- P $\neq$ NP and
- for a minimization problem only values with integers as target function values are possible and
- decision whether $v_{\text{opt}}(x) \leq k$ is NP-complete

then there is no approxmation algorithm with performance guarantee $< 1 + \frac{1}{k}$. This is a result for $(k, k+1)$-gap-problems.

We provide 2 examples for this theorem.

16.1.3 Example: BIN-Packing

Given. $a_1, \ldots, a_n \in (0, 1]$.
Find. Minimum number of bins, each bin has capacity 1.

No approximation algorithm with performance guarantee $< \frac{3}{2} = 1.5$ for Bin-packing exists if P $\neq$ NP. But there exists an asymptotical PTAS for BIN-Packing.

16.1.4 Example: Vertex coloring

Given. Undirected graph $G = (V, E)$.
Find. Coloring of V with minimum number of colors ($= \chi(G)$). Decide whether $\chi(G) \leq k$.

We can decide this in polynomial time for $k = 1$ and $k = 2$. For $k \geq 3$ this is an NP-complete problem ($(3, 4)$-gap-problem). Thus there is no approximation algorithm with performance guarantee $< \frac{4}{3}$ if P $\neq$ NP.

In comparison to BIN-Packing we cannot provide any better asymptotical results.

Theorem. If P $\neq$ NP there is no approximation algorithm for vertex coloring with asymptotical performance guarantee $c < \frac{4}{3}$.

Proof of the theorem. (expansion argument) We construct a graph $G_K = (V_K, E_K)$ and $\chi(G_K) = K \cdot \chi(G)$ with $K \in \mathbb{K}$.

Idea. $(3, 4)$-gap-problem for G becomes a $(3K, 4K)$-gap.

Consider K disjoint copies of G. We connect every vertex with all vertices in different copies. We have to show that $\chi(G_K) = K \cdot \chi(G)$.

- $\chi(G_K) \leq K \cdot \chi(G)$ can be ensured: Use in every copy a different set of colors.
- $\chi(G_K) \geq K \cdot \chi(G)$: We use $\chi(G)$ colors in G and because of the construction of G_K no color can occur in more than 1 copy.

Theorem. If P $\neq$ NP, NP-hard optimization problems with "small solutions" (ie. $\exists$ polynomial p: $\forall x \in S(x) : v(x, s) \in \mathbb{Z}_0^t$ and $v(x, s) \leq p(|x|)$ with $S(x)$ as the set of valid solutions and $v(x, s)$ as target function value) have no FPTAS.

Corollary. If P $\neq$ NP there does not exist a FPTAs for *strong* NP-hard problems.

Typical problems with "small solution values": CLIQUE, MAX-SAT, VERTEX COVER. And unlike KNAPSACK, PARTITION.

16.2 Construction at basis of PCP results

PCP theorem in our previous definition: NP = PCP($\log n$, 1). From this point of view we cannot find any relation to approximation algorithms.

Now consider MAX-3SAT.

Given. 3SAT equation ϕ
Find. Maximum number of satisfiable clauses.

Until 1992 it was an open question whether an ρ approximation algorithm exists for MAX-3-SAT for arbitrary $\rho > 1$. With the PCP results we gain: No, if $\mathrm{P} \neq \mathrm{NP}$.

The relation is that the PCP theorem has the following equivalent definition (PCP theorem in regards of hardness of approximation):

> There exists a constant $\hat{\rho} > 1$ such that all languages $L \in \mathrm{NP}$ there exists a polynomial time computable function f which maps strings to 3SAT equations such that
>
> $$x \in L \Rightarrow \mathrm{val}(f(x)) = 1$$
>
> $$x \notin L \Rightarrow \mathrm{val}(f(x)) < \hat{\rho}$$
>
> with $\mathrm{val}(x)$ as the quotient of maximum number of satisfied clauses and number of clauses. Distinguishing those cases is a gap-problem.

It immediately follows the following corollary.

Corollary. There exists a constant $\rho > 1$ such that if an ρ approximation algorithm for MAX-3-SAT exists, $\mathrm{P} = \mathrm{NP}$. Therefore approximation for MAX-3-SAT is not arbitrary. It can be close to 1.

With an ρ approximation algorithm A for MAX-3-SAT we get a polynomial algorithm for deciding whether $x \in L$.

Remark. The classical Cook-Levin idea is not applicable close to 1. We need the gap!

Now we want to discuss why both definitions of the PCP theorem are equivalent. Consider this generalization of 3SAT (CSP, constraint satisfaction problem):

Is $q \in \mathbb{N}$. An instance of q-CSP is a collection of functions $\phi_1, \dots, \phi_m$ (so-called "constraints") with $\phi_i : \{0,1\}^n \to \{0,1\}$ such that every ϕ_i depends on at most q input values.

Therefore for all $i \in \{1, \dots, m\}$ there exists $j_1, \dots, j_q \in \{1, \dots, n\}$ such that $\phi_i(u) = f(u_{j1}, \dots, u_{jq}) \forall u \in \{0,1\}^n$.

If $\phi_i(u) = 1$ we state that $u \in \{0,1\}^n$ satisfies the constraint ϕ_i (otherwise unsatisfiable).

The quotient of the number of satisfied constraints and all constraints is given by $\frac{\sum_{i=1}^m \phi_i(u)}{m}$. Denote $\mathrm{val}(\phi)$ as the maximum of all quotients. ϕ is called satisfiable if $\mathrm{val}(\phi) = 1$.

Remark. 3SAT is given with $q = 3$ and all constraints are disjunctions of literals.

Definition. (GAP-CSP) Is $q \in \mathbb{N}$ and is $\rho < 1$. Then we define ρ-GAP q-CSP as the problem to determine whether a given instance ϕ of q-CSP

- $\mathrm{val}(\phi) = 1$ (denote ϕ as Yes instance of ρ-GAP q-CSP)
- $\mathrm{val}(\phi) < \rho$ (denote ϕ as No instance of ρ-GAP q-CSP)

We state that ρ-GAP q-CSP is NP-hard for every language $L \in$ NP, if there is a polynomial time computable function f which maps strings to q-CSP instances such that

$$x \in L \Rightarrow \mathrm{val}(f(x)) = 1$$

$$x \notin L \Rightarrow \mathrm{val}(f(x)) < \rho$$

16.3 3 definitions of the PCP theorem

Now we have 3 equivalent definitions of the PCP theorem:

1. NP = $\mathrm{PCP}(\log \mathrm{n}, 1)$
2. The previous theorem in the statement above. A constant $g < I$ such that $\forall L \in$ NP there exists a polynomial computable function f : Strings $\rightarrow$ (Representation) $\vee$ 3SAT equations with

$$x \in L \Rightarrow \mathrm{val}(f(x)) = 1$$

$$x \notin L \Rightarrow \mathrm{val}(f(x)) < g$$

3. There exists a $q \in \mathbb{N}$ and $\rho \in (0, 1)$ such that ρ-GAP q-CSP is NP-hard.

16.3.1 The first definition implies the third

Is NP $\subseteq \mathrm{PCP}(\log \mathrm{n}, 1)$. We show that $\frac{1}{2}$-GAP q-CSP is NP-hard for appropriate constant.

It's enough that show only a single NP-complete language L (eg. reduce 3SAT to $\frac{1}{2}$-GAP q-CSP for appropriate constant q). From NP $\subseteq \mathrm{PCP}(\log \mathrm{n}, 1)$ it follows that there is a PCP verifier such that the verifier V has a constant number of proof bits q requested/verified and used $c \cdot \log n$ random bits (with c as constant). For given input x and random vector $r \in \{0, 1\}^{c \cdot \log n}$ define $V_{x,r}$ as the function returning 1 as output iff the proof B gets accepted by V.

Because q proof bits are relevant, $V_{x,r}$ only depends on q positions. Therefore $\forall x \in \{0,1\}^n$ returns the collection

$$\phi = \{V_{x,r}\}_{r \in \{0,1\}^{c \cdot \log n}}$$

a q-CSP instance of polynomial size.

Because V has polynomial runtime the transformation of x to ϕ has also polynomial runtime. Because V is our PCP verifier, for $x \in$ 3SAT (x must be satisfiable) its given that $\phi = 1$ and for $x \notin$ 3SAT, val$(\phi) \leq \frac{1}{2}$. Therefore $\frac{1}{2}$-GAP q-CSP NP-hard.

16.4 The third definition implies the first

Assume ρ-GAP q-CSP is NP-complete for constant $q \in \mathbb{N}, \rho \in (0,1)$. This can be translated to PCP-system with q proof bits, logarithmical number of random bits and error probability $\leq \rho$ for every language L:

Given input x. The verifier calls the reduction $f(x)$ to gain the q-CSP instance $\phi = \{\phi_i\}_{i=1,\dots,m}$.

It is expected that the proof B is an assignment of values to the variables in ϕ. Verification: Select randomly $i \in \{1,\dots,m\}$ and check whether ϕ_i is satisfied (this involves q position requests). Obviously the verifier accepts with probability 1 if $x \in L$. If $x \notin L$, acceptance with probability $\leq \rho$.

This error probability can be improved to $\frac{1}{2}$ with costs of a constant factor in regards to the number of random bits and the number of proof positions.

16.5 The second definition equates the third definition

Because 3SAT is special case of ρ-CSP, it immediately follows that the second definition implies the third. We have to show that the third definition implies the second.

Is $\varepsilon > 0$ and $q \in \mathbb{N}$ such that $(1-\varepsilon)$-GAP q-CSP is NP-hard ($q, \varepsilon \exists$ because of the third definition).

Is $\{\phi_i\}$ a q-CSP instance with n variables and m constraints.

Each constaint ϕ_i can be represented as conjunction of $\leq 2^q$ SAT clauses (thus each clause is an disjunction of $\leq q$ literals).

Is ϕ' the collection of $\leq m \cdot 2^{\phi}$ clauses resulting from all $\phi_i, i \in \{1,\dots,m\}$. If ϕ is a Yes instance of $(1-\varepsilon)$-GAP q-CSP (ie. ϕ is satisfiable), then there exists a truth assignment such that all clauses of ϕ' are satisfied. If ϕ is a No instance of $(1-\varepsilon)$-GAP q-CSP then at least the portion ε of constraints of ϕ is violated in each truth assignment.

At least a portion of $\frac{\varepsilon}{2^q}$ of constraints of ϕ' is violated. Use proof technique for theorem of Cook to transform each clause C to q variables $u_1, \dots, u_q$ in a set of clauses $C_1, \dots, C_q$ to the variables $u_1, \dots, u_q$ and additional temporary variables $y_1, \dots, y_q$ such that

- every clause C_j is a 3SAT clause
- if $u_1, \dots, u_q$ is selected such that C is satisfied, then there exists an assignment of $y_1, \dots, y_q$ such that $(u_1, \dots, u_q, y_1, \dots, y_q)$ satisfies the clauses $c_1, \dots, c_q$.
- if $u_1, \dots, u_q$ such that C is not satisfied, then for all assignments of $y_1, \dots, y_q$ there is one clause C_i such that C_i is not satisfied by $u_1, \dots, u_q, y_i, \dots, y_q$.

From ϕ' generate ϕ'' as follows: Apply the technique from above to all clauses of ϕ'; we get a collection of $\leq q \cdot m \cdot 2^q$ clauses.

ϕ'' is a 3SAT instance. In total $\phi \to \phi''$.

- If ϕ is satisfiable, then ϕ and ϕ'' are satisfiable.
- If ϕ' is unsatisfiable, then every assignment of ϕ violates at least a portion ε of constraints. This corresponds to a $\frac{\varepsilon}{2^q}$ portion of ϕ' and $\frac{\varepsilon}{q \cdot 2^q}$ portion of constraints of ϕ''.

Let A be the version showing PCP theorem's hardness from approximation's point of view (definition 1). Let B be the second definition and C the third.

PCP verifier For A, it is PCP-verifier V. For B it is an CSP instance ϕ.

PCP proof For A, it is proof B. For B is it assignment of variables in ϕ.

Length of proof Not given for A. For B it is the number of variables.

Number of position lookups (q) For B and C, it is the maximum number of variables per constraint.

Number of random bits (r) For B and C, it is the logarithm of the constraint number m.

Error probability (typically $\frac{1}{2}$) For B and C, it is the maximum for val(ϕ) for a No instance.

We now know that (if P $\neq$ NP) MAX-3-SAT cannot be approximated as close to q as we wish (NP-hard problem).

But no particular value $\widetilde{g}$ follows from the ideas above such that $\widetilde{q}$-approximation of MAX-3-SAT is NP-hard.

Hastad was able (with an improved version of the PCP theorem) to show for 3SAT that for all $\varepsilon > 0$ if there exists a $(\frac{8}{7} - \varepsilon)$-approximation of MAX-3-SAT, $\mathrm{P} \neq \mathrm{NP}$ follows.

17 Further negative approximation results

Other approximation results that follow from the second PCP-theorem definition. So far we have considered MAX-3-SAT and CSP.

Consider VERTEX COVER (minimum cardinality) and INDEPENDENT SET (maximum cardinality, corresponds to STABLE SET).

Theorem. There exists a constant $\gamma > 1$ such that γ-approximation for VERTEX COVER is NP-hard. For all $\beta > 1$, a β-approximation of INDEPENDENT SET is NP-hard.

The second theorem is stronger than the first theorem ("for all"). VERTEXCOVER $\in$ APX (compare with 2-approximation algorithm from start of this chapter).

17.1 INDEPENDENT SET

$$\text{CLIQUE} \notin \text{APX} \quad \text{if } \mathrm{P} \neq \mathrm{NP}$$

INDEPENDENT SET and VERTEX COVER are equivalent in regards of the corresponding optimization problem, but no statement (or equivalence) for approximation follows from this.

In a VERTEX COVER all vertices are covered (complement of VERTEX COVER is an INDEPENDENT SET).

Is N_{VC} the minimal cardinality of a VERTEX COVER and is N_{IS} the maximal cardinality of an INDEPENDENT SET. Is n the number of vertices in G. Then:

$$N_{VC} = n - N_{IS}$$

A β-approximation for INDEPENDENT SET returns INDEPENDENT SET of size $\frac{1}{\beta}N_{IS}$. If you want to use this for VERTEX COVER, we get VERTEX COVER with cardinality $n - \frac{1}{\beta}N_{IS}$. This provide us approximation guarantee:

$$\frac{n - \frac{1}{\beta}N_{IS}}{n - N_{IS}}$$

This can be arbitrarily bad, especially if IS is close to n. This is no surprise if we consider the last theorem. We want to provide a proof for this theorem.

17.2 Proof the last theorem

First we show (using the PCP theorem) for INDEPENDENT SET and VERTEX COVER that there is some constant $\rho > 1$ such that ρ-approximation is NP-hard.

Lemma. There exists a polynomially computable transformation f of 3SAT equations to graphs such that for every 3SAT equation ϕ, $f(\phi)$ is an undirected graph with n vertices which largest independent set has cardinality $\mathrm{val}(\phi) \cdot \frac{n}{7}$.

Proof. The proof follows from the classical NP-completeness reduction of INDEPENDENT SET.

Corollary. If $P \neq NP$ there exists constants $\rho, \rho' > 1$ such INDEPENDENT SET is not ρ-approximable and VERTEX COVER is not ρ'-approximable.

Proof. Is L a language in NP. From the second PCP-theorem definition we know that decision version of L is reducible to approximation of MAX-3-SAT Reduction provides a 3SAT equation ϕ which is either satisfiable $\mathrm{val}(\phi) = 1$ or $\mathrm{val}(\phi) < \rho$ with $\rho < 1$. Now we use the reduction from the Lemma:

$$\frac{1}{\rho} \text{ approximation of INDEPENDENT SET} \rightarrow \frac{1}{\rho} \text{ approximation of MAX-3-SAT}$$

This is a statement thatn $\frac{1}{\rho}$-approximation ($\rho' > 1$) for INDEPENDENT SET is NP-hard.

Regarding VERTEX COVER: Minimal vertex cover which can be derived from graph with the reduction has cardinality $n - \mathrm{val}(\phi)\frac{n}{7}$. So if VERTEX COVER has a ρ'' approximation with $\rho'' = \frac{7-\frac{1}{\rho}}{6}$ then we could find a vertex cover with size $\rho' \cdot (n - \frac{n}{7})$ in case $\mathrm{val}(\phi) = 1$ and this is $\leq n - \rho \cdot \frac{h}{7}$. We could distinguish between $\mathrm{val}(\phi) = 1$ and $\mathrm{val}(\phi) < \rho$ which is NP-hard.

Now we proof the theorem. Given is a graph G with n vertices. Consider graph G^k $\binom{n}{k}$ vertices correspond to k-sized subsets of V. Two subsets S_1 and S_2 are connected by an edge if $S_1 \cup S_2$ is an independent set in G.

The largest independent set in G^k corresponds to all k-sized subsets of the largest independent set in G and has therefore cardinality $\binom{N_{IS}}{k}$. We now take the graph from the reduction in the corollary above and construct the product with k. The quotient from the size of the largest independent set in the two cases is $\frac{\binom{N_{IS}}{k}}{\binom{\rho \cdot N_{IS}}{k}}$. This is an approximation of about $\left(\frac{1}{\rho}\right)^k$. If k is large enough, then the approximation guarantee is arbitrarily bad. The runtime is therefore n^k; polynomial.

In the following lectures we want to develop a reduction term for approximation.

18 Reduction of approximation problems

We need an appropriate reduction term for approximation.

There are a number of terms available. We discuss PTAS-reductions.

$$\begin{tikzcd}
\text{Problem A} & \text{Problem B} \\
\text{Input}\quad x \arrow[r, bend right, "f"] & f(x) = x' \arrow[d, bend left, "\text{approximation algorithm for B}"] \\
S = g(s') \\ S \in S_A(x) & s' \in S_B(x') \arrow[l, bend left, "g"'] \\
& \text{valid solution for B}
\end{tikzcd}$$

We need an appropriate "translation" of approximation guarantee s' in regards of B to s in regards of A.

Especially for PTAS reduction we want to insert a PTAS for B and we require

$$r_B(x', s') \leq 1 + \alpha(\varepsilon) \Rightarrow r_A(x, s) \leq 1 + \varepsilon$$

Definition. A PTAS reduction of problem A to problem B ($A \leq_{\text{PTAS}} B$) is denominated by a triple (f, g, α) with

- For input x of A, input $f(x)$ for B is computable in polynomial time.
- For input x of A and solution $s' \in S_B(f(x))$ and $\varepsilon \subseteq Q^t$, $g(x, s', e)$ can be computed in polynomial time.
- $\alpha : Q^+ \to Q^+$ is surjective, polynomially computable and $r_B(f(x), s') \leq 1 + \alpha(\varepsilon) \Rightarrow r_A(x, g(x, s', \varepsilon)) \leq 1 + \varepsilon$.

Lemma. $A \leq_{\mathrm{PTAS}} B, B \in \mathrm{PTAS} \Rightarrow \mathrm{A} \in \mathrm{PTAS}$.

Proof. We need a PTAS for A (input x, ε). Apply PTAS to B with input $f(x)$ and guarantee $1+\alpha(\varepsilon)$. We get solution s' for B with $r_B(f(x), s') \leq 1+\alpha(\varepsilon)$. Return $s = g(x, s', \varepsilon)$. From the third property of the definition we know that

$$r_A(x, s) \leq 1 + \varepsilon$$

Further properties of $\leq_{\mathrm{PTAS}}$:

- $\leq_{\mathrm{PTAS}}$ is reflective.
  $(f, g, \alpha = \mathrm{id})$.
- $\leq_{\mathrm{PTAS}}$ is transitive.

$\leq_{\mathrm{PTAS}}$ is of partial order. We can define $=_{\mathrm{PTAS}}$ by $A =_{\mathrm{PTAS}} B \Leftrightarrow A \leq_{\mathrm{PTAS}} B$ and $B \leq_{\mathrm{PTAS}} A$.

Example. P_1: Determine clique with maximum number of vertices. P_2: Determine an independent set of maximum number of vertices. $P_1 =_{\mathrm{PTAS}} P_2$.

Theorem. $\mathrm{MAX}-3-\mathrm{SAT} \leq_{\mathrm{PTAS}} \mathrm{CLIQUE}$.

Proof. Consider the classical reduction. 3SAT equation ϕ with clauses $C_1, \dots, C_m$. 3SAT $\leq_{\mathrm{p}}$ CLIQUE.

$f(\phi) = G$ with $G = (V, E)$. In the classical reduction we use, that ϕ is satisfiable $\Leftrightarrow f(\phi)$ has clique of cardinality m.

We now use an observation: At least k clauses are satisfiable $\Leftrightarrow \exists$ clique with cardinality $\geq k$.

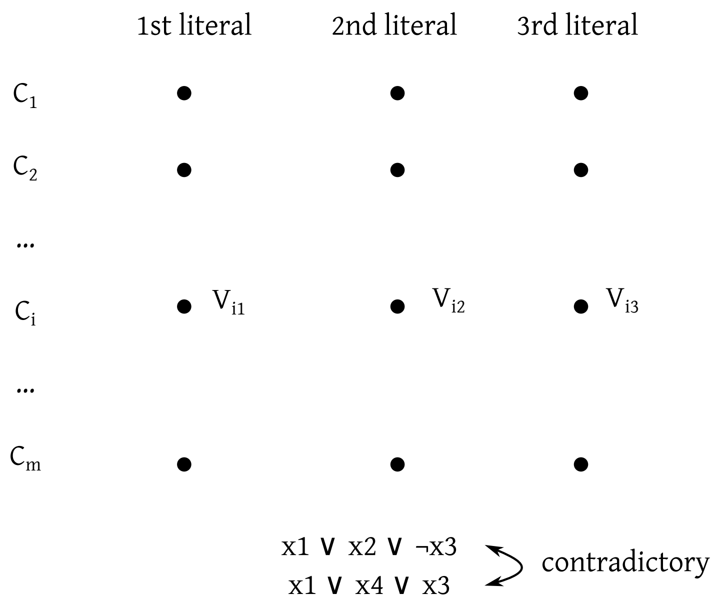
Edge are introduced between vertices in different rows, which do not correspond to contradictory clauses $\rightarrow$ Clique with k vertices.

Let clauses $C_{i1}, \dots, C_{ik}$ be (simultaneously) satisfiable.

For rows $i_1, \dots, i_k$ there exists a satisfiable literal. The corresponding vertices are pairwise connected by edges (in different rows and are not contradictory) $\rightarrow$ clique with k vertices.

Assume $V_{i1,j1}, \dots, V_{ik,jk}$ create a clique with k vertices. Literals at position (i_1, j_1) (j_1-th literal in clause C_{i1}) $\dots (i_k, j_k)$ is satisfiable. Choose a truth assignment which is arbitrarily consistent with the literal assignment $\rightarrow$ k clauses are satisfied.

From input ϕ and clique V' with $|V'| = k$ in $G = f(\phi)$ the variable assignment $g(\phi, V')$



Remark. Not every classical reduction is appropriate for approximation reduction.

Example. MAX − 4 − SAT → MAX − 3 − SAT.

4-SAT equation ϕ with m clauses. Clause $C_i = (l_{i1} \vee l_{i2} \vee l_{i3} \vee l_{i4})$. This is represented as 2 3SAT clauses

$$(l_{i1} \vee l_{i2} \vee z_i) \wedge (l_{i3} \vee l_{i4} \vee \neg z_i)$$

z_i is a new variable. We get a 3SAT equation ϕ' with $2m$ clauses.

Problem. Here it is trivialy $\geq$ half of clauses in ϕ' is satisfiable. So this is inappropriate for PTAS reduction.

Next goal. (completeness term) Introduce class NPO as abstract class for optimization problem analogously to NP for decision problems.

Definition. (NPO) An optimization problem A corresponds to class NPO, if

- for all x and $s(x)$ the target function value $v(x, s)$ is an integer (could be relaxed to Q).
- for all x and s, the test whether $s \in S(x)$ (s is valid) must be computable in polynomial time.
- $v(x, s) \forall x, s$ is computable in polynomial time.

MIN-NPO	Minimization problem in NPO
MAX-NPO	Maximization problem in NPO

Definition. An optimization problem A is NPO-complete iff

- $A \in$ NPO
- $\forall B \in$ NPO : B $\leq_{\text{PTAS}}$ A

In the following we consider an optimization problem of NPO. Therefore we are interested for APX, PTAS, ... now the intersection of NPO with the classes defined previously.

The completeness definition for PTAS and APX is analogous.

There are MAX-NPO and MIN-NPO complete problems. It can be proven that MAX-W-SAT is MAX-NPO complete and MIN-W-SAT is MIN-NPO complete.

Given. SAT equation ϕ with variables $x_1, \ldots, x_n$ with clauses $C_1, \ldots, C_n$. Weights $w_1, \ldots, w_n \in \mathbb{N}_0$.

Target function value.

$$v(a) = \begin{cases} \max 1, \sum_{i=1}^{n} w_i \cdot a_i & a \text{ is satisfiable} \\ 1 & \text{else} \end{cases}$$

$a \in \{0, 1\}^n$ truth assignments.

Furthermore we can show that MAX-W-SAT $=_{\text{PTAS}}$ MIN-W-SAT. MAX-W-SAT and MIN-W-SAT are NPO-complete.

Another complexity class in relation with approximation is MAX-SNP (concept of Paradimitriou and Yannakakis).

Definition. (Strict NP) All properties of the structure

$$\exists s \forall x_1 \forall x_2, \ldots, \forall x_n : \Phi(S, G, x_1, \ldots, x_n)$$

where Φ is a quantorfree expression of first order, which contains the variables $x_1, \ldots, x_n$ and contains graph structure G and S.

Step towards optimization problem MAX-SNP:

$$\max_S |\{(x_1, \ldots, x_n) \in V^n | \Phi(G_1, G_2, \ldots, G_m, S, x_1, \ldots, x_n)\}|$$

where $G_1, \ldots G_m$ is input relation over a finite set V.

Find. Relation S such that the number of n-tuples with Φ is satisfied.

18.0.1 Example 1

Given. Graph $G = (V, E)$.
Find. Cut which cuts the most number of edges.

Relation G (which also corresponds to the graph) describes set of edges

$$\max_{S \subseteq V} |\{(x, y) : G(x, y) \vee G(y, x) \wedge S(x) \wedge \neg S(y)\}|$$

(meaning there exists an edge between x and y and and an intersecting edge exists).

18.0.2 Example 2: MAX-2-SAT

3 input relations: G_0, G_1, G_2. G_i describes all clauses with i intersected literals.

$$G_0(x, y) : x \vee y \text{ is clause}$$
$$G_1(x, y) : \neg x \vee y \text{ is clause}$$
$$G_2(x, y) : \neg x \vee \neg y \text{ is clause}$$

S represents a variable with truth value True.

$$\max_{S \subseteq V} |\{(x, y) : \Phi(G_0, G_1, G_2, S, x, y)\}|$$

$\Phi : [G_0(x, y) \wedge (S(x) \vee S(y))] \vee [G_1(x, y) \wedge (\neg s(x) \vee s(y))] \vee [G_2(x, y) \wedge (\neg s(x) \vee \neg s(y))]$

18.0.3 Example 3: MAX

INDEPENDENT SET in graph G with restricted degree k.

Representation of G as $(k + 1)$-ary relation H. H contains $|V|$ $(k + 1)$-tuples.

$(y_1, \ldots, y_k)$ where y_i neighbors of x are repeated vertices, if they are of degree $x < k$.

$$\max_{S \subseteq V} |\{(x, y_1, \ldots, y_k) : (x, y_1, \ldots, y_k) \in H \wedge x \in S \wedge y_i \notin S, i \in \{1, \ldots, K\}\}|$$

S is an independent set.

Remark. We can show the following.

- Closure of MAX-SNP under PTAS reduction is APX.
- Every problem in MAX-SNP has an approximation algorithm with constant guarantee.
- INDEPENDENT SET $\notin$ MAX-SNP, if P $\neq$ NP.

19 Positive results

19.1 Linear-integer optimization models ("rounding")

19.1.1 MAX-SAT

Given. SAT equation ϕ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.

For clause C_i denominate V_i^+ the set of non-negative variables in C_i. V_i^- is the set of negative variables in C_i.

We have 2 groups of variables:

1. x is variable. $x_i \in \{0, 1\}$. $x_i = 1$ if x_i is true in ϕ. Else $x_i = 0$.
2. z is variable. $z_j \in \{0, 1\}$. $z_j = 1$ if clause C_j is satisfied. Else $z_j = 0$.

The number of satisfied clauses is

$$\max \sum_{j=1}^{m} z_j$$

We require (IP$_{\text{MS}}$):

$$\sum_{x_j \in V_j^+} x_i + \sum_{x_i \in V_j^-} (1 - x_i) \geq z_j \qquad j = 1, \ldots, m$$

with

$$x_i \in \{0, 1\}, i = 1, \ldots, n \qquad z_j \in \{0, 1\}, j = 1, \ldots, m$$

This requires integers.

Linear programming relaxation (LP$_{\text{MS}}$) Instead we define "with":

$$0 \leq x_i \leq 1$$