This document contains some basics of the complexity theory. It is mostly based on the lecture course delivered at CS dept. by Meran G. Furugyan. Differences are subtle. Disclaimer of warranties apply: if you fail an exam because of an error on my part, it's your problem – you could attend lectures after all. However, major bugs have some chances to be fixed and can be reported to `<ghost@cs.msu.su>`, and new versions will probably appear at `http://chronos.cs.msu.su/~ghost/MyDocuments/Text/complexity.pdf`

It is meant that information given here be more usable than in scanned lecture course, but you should read the text *before* the exam – you can be burned otherwise, as some parts are, unfortunately, obscure, and others may contain mistakes of various kinds.

Beware! It is very *alpha version.*

Contents

1 NP-complete problems

Due to limited author's time, definitions of Turing machine and NP-completeness are omitted. They can be found in last year's lectures that are available somewhere in the net.

1.1 Basic NP-complete problems

[There should go tree]

Problem 1 (SAT) *Given a CNF $E = E_1 \dots E_n$, find if it's satisfiable*

Problem 2 (SAT-3) *Given a CNF $E = E_1 \dots E_n$, where each E_k has the form $E_k = v_{k1} \vee v_{k2} \vee v_{k3}$, find if it's satisfiable*

Problem 3 (3DM) *Given three sets X, Y, Z of equal cardinality N and a set $W \subseteq X \times Y \times Z$, find if there's $W' \subseteq W$ such that $|W'| = N$ and no pair of elements in W' agree in any coordinate. That is, each element from X, Y and Z occurs in exactly one triad from W'.*

Note: This is very similar to 2-dimensional matching: given two equally sized groups of boys and girls, and a set of possible pairs, pair 'em all. Complexities, however, are *very* different.

Problem 4 (Partition) *Given a set $A = (a_1, \dots, a_n)$ of numbers, is there a partition of A into A' and A'' ($A' \cup A'' = A$, $A' \cap A'' = \emptyset$) such that $\sum_{a_i \in A'} a_i = \sum_{a_i \in A''} a_i$.*

Problem 5 (Vertex cover) *Given an undirected graph $G = (V, A)$ and a number K determine if there's $V' \subseteq V$ such that $|V'| \leqslant K$ and $\forall (u, v) \in A \; u \in V' \vee v \in V'$*

Problem 6 (Independent Set) *Given an undirected graph $G = (V, A)$ and a number K determine if there's $V' \subseteq V$ such that $|V'| \geqslant K$ and $\forall u, v \in V'$ u and v are not adjacent in G.*

Problem 7 (Clique) *Given an undirected graph $G = (V, A)$ and a number K determine if there's $V' \subseteq V$ such that $|V'| \geqslant K$ and subgraph induced by V' is complete.*[1]

Problem 8 (Hamiltonian circuit) *Given an undirected graph $G = (V, A)$ find if there's a simple circuit*[2] *passing through all the vertices in the graph.*

Theorem 1 *SAT is in $\mathbb{NPC}$.*

Proof. The fact that SAT $\in \mathbb{NP}$ is considered obvious. The aim is the prove that $\forall L \in \mathbb{NP}$, $L \leqslant_p L(SAT)$. From the definition of $\mathbb{NP}$ we know that exists NDMT M accepting L, and $\exists p$ such that time bound is $p(n)$, where n is string length. That is, exists a computation of M with less than $p(n)$ steps that accepts that string. Existence of such computation can be formulated in terms of boolean expressions.

Let's assume that NDMT $(Q, q_0, q_y, q_N, \Sigma, \Gamma, \epsilon \in \Gamma \setminus \Sigma, \delta)$ have such numbering of states and symbols: $Q = \{q_0, q_1 = q_Y, q_2 = q_N, \dots, q_r\}$, $\Gamma = \{S_0 = \epsilon, \dots, S_t\}$. Input string will be denoted as $(l_1, \dots, l_n)$.

The formulation used three groups of variables.

i. Q_{ik} – is machine in time i in state k.

ii. H_{ij} – is machine in time i look at position j.

iii. S_{ijl} – is machine in time i in position j contain symbol l.

Indices are $i = [0, p(n)]$, $j = [-p(n), p(n) + 1]$, $k = [0, r]$, $l = [0, t]$.

[1] Subgraph induced by V' is $(V', E') | E' = \{(x, y) \in E, x \in V', y \in V'\}$.

[2] i.e. circuit that has no duplicate vertices

Six groups of conjuncts are used.
C1. Assures that machine is in one state at a moment.

i. $(Q_{i,0} \vee \ldots \vee Q_{i,r})$.
ii. $\overline{Q_{i,k}} \vee \overline{Q_{i,k'}}$, $\forall k, k' : 0 \leqslant k < k' \leqslant r$.

C2. Assures that position of head is determined.

i. $(H_{i,-p(n)} \vee \ldots \vee H_{i,p(n)+1})$.
ii. $\overline{H_{i,j}} \vee \overline{H_{i,j'}}$, $\forall j, j' : -p(n) \leqslant k < k' \leqslant p(n)+1$.

C3. Assures only one symbol in a position.

i. $(S_{i,j,0} \vee \ldots \vee S_{i,j,t})$
ii. $\overline{S_{i,j,l}} \vee \overline{S_{i,j,l'}}$, $\forall k, k' : 0 \leqslant l < l' \leqslant t$.

C4. Sets initial configuration.

i. $S_{0,1,l_1}, \ldots, S_{0,n,l_n}$.
ii. $S_{0,1,0}, S_{0,n+1,0}, \ldots, S_{0,p(n)+1,0}$.
iii. $Q_{0,0}$.
iv. $H_{0,1}$.

C5. Makes sure accepting state is reached.

i. $Q_{p(n),1}$

C6. Guarantees that steps are performed correctly.[3]

i. $(H_{i,j} \vee \overline{S_{i,j,l}}) \vee S_{i+1,j,l}$ – positions that are not observed are not changed.
ii. $\forall (q_{k'}, s_{l'}, \Delta) = \delta(q_k, s_l)$ $\begin{cases} (\overline{Q_{ik}} \vee \overline{H_{ij}} \vee \overline{S_{ijl}}) \vee Q_{i+1,k'} \\ (\overline{Q_{ik}} \vee \overline{H_{ij}} \vee \overline{S_{ijl}}) \vee S_{i+1,l'} \\ (\overline{Q_{ik}} \vee \overline{H_{ij}} \vee \overline{S_{ijl}}) \vee H_{i+1,j+\Delta} \end{cases}$
iii. If $q_k \in \{q_Y, q_N\}$ use the conjunct given above, but let $k' = k$ and $l' = l$.

Theorem 2 *3-SAT is in* $\mathbb{NPC}$

Proof of this theorem can be found elsewhere.

Theorem 3 *Vertex Cover, Independent Set and Clique are in* $\mathbb{NPC}$.

Proof. Let us note that the following statements are equivalent:

- V' is vertex cover.
- $V \setminus V'$ is independent set.
- $V \setminus V'$ is a clique in the complement of (V, A).[4]

Thus, it's enough to prove NP-completeness of one of these problems, for instance, independent set. It is done by reduction from 3-SAT. For $E = E_1 \ldots E_n$ build a graph by the following rules:

- For each variable x_i, make two vertices x_i and $\overline{x_i}$. Add an edge between them.
- For each conjunct E_k, make vertices corresponding to each of the three literals in the conjunct. Link all three vertices together, making an triangle.

[3] Recall that $x \Rightarrow y$ can be written $!x \vee y$. Left part of implication will be parenthesized below.
[4] Complement of (V, A) is $(V, \{(u, v) : u \in V, v \in V (u, v) \notin A\})$.

- Literal is either x_i or $\overline{x_i}$. Vertex for the literal should be joined with vertex corresponding to either x_i or $\overline{x_i}$, accordingly.

Each variable can yield no more than one vertex in independent set. Each literal similar gives no more than one vertex. Therefore, cardinality of independent set cannot exceed $n+p$. It turns out that if this number is reached, the CNF is satisfiable.

If E is satisfiable, independent set can be constructed thusly.

- For all variables, append vertex x_i to the independent set, if $x_i = false$ and append $\overline{x_i}$ otherwise.
- For each E_k some of it's literals is true. Since by construction variable vertex, joined with that literal is not included in the independent set, the literal can be added.

This way, independent set of cardinality $n+p$ is created.

Proof in the other direction is similar. Reader is expected to devise it in much less time than required to typeset it.

1.2 More NP-complete problems

Problem 9 (Interrupt-free scheduling) *Given a number of tasks N, theirs durations τ_i, number of processors m and time limit T, determine, whether exists an interrupt-free schedule not exceeding the specified time*

Theorem 4 *Interrupt-free scheduling is in $\mathbb{NPC}$.*

Proof. Reduction from partition. For the set A create $N = |A|$ tasks with $\tau_i = a_i$, let $m = 2$, and $T = \frac{1}{2}\sum a_i$.

Problem 10 (Ordering within interval) *Given a number of tasks N, theirs durations τ_i, allowed time ranges (b_i, f_i) and time limit T, determine whether exists an interrupt-free schedule on one processor not exceeding the specified time.*

Theorem 5 *Ordering within interval is in $\mathbb{NPC}$.*

Proof. Reduction from partition. For the set A create $N = |A|$ tasks with $\tau_i = a_i$, let $m = 2$ and $T = 1 + \frac{1}{2}\sum a_i$. Create an auxiliary task with duration of 1 and fix it in the middle by setting time range to $(B/2, B/2+1)$. For all the other tasks set time range to $(0, T)$.

Problem 11 (Hitting set) *Given a set S and a collection $C_1, \ldots, C_n$, $C_i \subseteq S$ and number K, is there exist $S' \subseteq S : |S'| < K, \forall C_i \exists x \in S' : x \in C_i$.*

Theorem 6 *Hitting set is in $\mathbb{NPC}$.*

Proof. Reduction from vertex cover. For graph (V,A), let $S = V$ and $\forall (u, v) \in A$ append set $\{u, v\}$ to the collection.

Problem 12 (Set packing) *Given a collection C of finite sets, and number K, are there K elements of C, no pair of which intersect.*

Theorem 7 *Set packing is in $\mathbb{NPC}$.*

Proof. Reduction from 3DM. If sets X, Y, Z do not intersect, then elements of W can be treated simply as (unordered) sets, and reduction is trivial. Otherwise, renumber elements in Y and Z.

Problem 13 (Subgraph isomorphism) *Find if graph G_1 contains subgraph isomorphic to graph G_2*

Theorem 8 *Subgraph isomorphism is in* $\mathbb{NPC}$.

Proof. Take complete graph with K vertices as G_2, and see that clique is reducible to subgraph isomorphism.

Problem 14 (Bounded degree spanning tree) *Given a graph $G = (V, A)$ and a number K find if there exists spanning tree such that degree of every vertex if less than K.*

Theorem 9 *Bounded degree spanning tree is in NPC.*

Proof. Hamiltonian path is actually a spanning tree with degree less than two (or one)[5].

Problem 15 (Knapsack) *Having n objects with volumes v_i and prices s_i, and a knapsack of volume V, can we select some objects (find $N' \subseteq \{1, \ldots, n\}$) so that they can be placed ($\sum_{i \in N'} v_i < V$) yet their price is more that some given limit S, ($\sum_{i \in N'} x_i > S$).*

Theorem 10 *Guess what... Those who will can demand a $123.3i award* ☺

Proof. Reduction from partition. For a set A create $|A|$ objects set $v_i = a_i$, $s_i = a_i$, $V = S = \sum(a_i \in A)/2$.

Problem 16 (Longest path in graph) *Is there a simple path in graph (V, A) passing through more than K vertices?*

Reduction from hamiltonian circuit. Trivial. **Note.** For oriented graph without circuits this problem is polynomial, despite the fact that number of simple paths can be exponential.

Problem 17 (Largest common subgraph) *Find, is graphs G_1 and G_2 have some isomorphic subgraphs with more that K vertices.*

Subgraph isomorphism trivially reduces to this problem.

Problem 18 (Minimum Sum of Squares) *Given a set $(a_1, \ldots, a_n)$ and integers J, K, can A be partitioned into K disjoint sets so that $\sum_{i=1}^{K} (\sum A_k)^2 \leqslant J$.*

Reduction from partition. Let $K = 2$ and $J = \frac{B^2}{2}$. If partition is possible, then $(\frac{B}{2})^2 + (\frac{B}{2})^2 = \frac{B^2}{2} = J$. If partitions is not possible then for every A_1 and A_2 $\sum A_1 = \frac{B}{2} - \epsilon$ where $\epsilon > 0$. The sum them will be $\frac{B^2}{2} + 2 * \epsilon^2$.

Problem 19 (Late tasks weight minimization) *Given set of N tasks, with executions times τ_i, allowed ranges $[b_i, f_i]$ and weights w_i, is it possible to create schedule for one processors so that sum of weights of all late tasks is less that K?*

Reduction from partition. Let $\tau_i = w_i = a_i$, $b_i = 0$, $f_i = B/2$, $K = B/2$.

Problem 20 (Bin packing) *Having an infinite number of bins, each of volume V, can we put set of N objects with volumes v_i into less than K bins.*

This problem is extremely similar to scheduling. The only difference seems to be in variable names.

Problem 21 (Cosine product integral) *Given numbers $a_1, \ldots, a_n$, is it true that $\int_0^{2\pi} [\cos(a_i x)] dx \neq 0$.*

1.

$$\int_0^{2\pi} \cos(ax) dx = \begin{cases} 2\pi & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

[5] It depends on whether the graph is directed or not.

2.

$$cos(a_1x)cos(a_2x) = \frac{cos(a_1+a_2)x + cos(a_1-a_2)x}{2}$$

$$cos(a_1x)cos(a_2x)cos(a_3x) = \frac{\begin{array}{c} cos(a_1+a_2+a_3)x + cos(a_1+a_2-a_3)x + \\ +cos(a_1-a_2+a_3)x + cos(a_1-a_2-a_3)x \end{array}}{4}$$

3. From the formulae above, we conclude that product of cosines is the sum of 2^{n-1} cosines, and coefficients in those cosines correspond to every possible partition of $\{a_1, \ldots, a_n\}$. So, $\int \neq 0$ iff exists coefficient equal to zero, i.e. iff answer to partition problem is "yes".

Problem 22 (Dominating set) *Dominating set in graph $G = (V, A)$ is $V' \subseteq V : \forall v \in V \setminus V' \exists u \in V' : (u, v) \in A$ The question is whether is a given graph exists dominating set with size less than K.*

Reduction from vertex cover. Confine discussion to graphs without isolated vertices. Note that for such graphs *any* vertex cover is also a dominating set. The opposite is false: consider a triangle: any single vertex is dominating set but not vertex cover. It is possible to force at least one end vertex of each edge to be included to dominating set.

Graph G' is made by creating, for each edge, an auxiliary vertex and joining it with both endpoints. Vertex cover in original graph will be dominating set in G' still. On the other side, every dominating set should contain, for each edge, either one of it's endpoints or the auxiliary vertex for the edge. If the former case apply to all the edges, we have vertex cover in original graph. Otherwise, it is possible to remove auxiliary vertex from dominating set and add one of the endpoints to it, still having dominating set.

Problem 23 (Ordering with minimum delay(?)) *Given set of N tasks, with executions times τ_i and allowed ranges $[b_i, f_i]$, is it possible to create schedule for one processor so that number of late tasks is less that K?*

Unlike with late tasks weight minimization, reduction from partition is not possible directly. We could try to create τ_i tasks for original task i, but such a reduction wouldn't be polynomial.

Take clique problem with graph $G = (V, E)$ ($|V| = n, |E| = m$) and number J. Let's use the obvious fact that if we have $J(J-1)/2$ edges, the number of theirs endpoints is not less than J and equal only in the case of a complete graph. New problem is:

i. $N = V \cap E$.

ii. $K = m - \frac{J(J-1)}{2}$.

iii. $\forall i \in V f_i = n + m$.

iv. $\forall i \in E f_i = \frac{J(J+1)}{2}$.

v. $\forall (i, j) \in E\ i \prec (i, j)\ j \prec (i, j)$.

So, it is required that at least $\frac{J(J-1)}{2}$ edges are scheduled in first $\frac{J(J+1)}{2}$ time slots. It is possible iff exists clique of J vertices. In this case we can schedule all those nodes and then edges.

2 Contraction of NP-complete problems

2.1 Pseudopolynomial algorithms

Consider problem Π. For each $I \in \Pi$, in addition to length function $l(I)$, define maximum function $M(I)$, as maximum number used in problem instance. Two pairs (l_1, M_1) and (l_2, M_2) are said to be polynomially equivalent, if two condition hold:

1. $\exists p_1, p_2 : l_1(I) \leqslant p_2(l_2(I)),\ l_2(I) \leqslant p_1(l_1(I))$
2. $\exists q_1, q_2 : M_1(I) \leqslant q_2(M_2(I)),\ M_2(I) \leqslant q_1(M_1(I))$

Such pairs can be used interchangeably.

Definition 1 *Algorithm A is pseudo-polynomial if $\exists p$ such that algorithm's complexity is bounded by $p(l(I), M(I))$.*

Definition 2 *Problem Π is problem with numeric parameters if $\nexists p : M(I) \leqslant p(l(I)),\ \forall I \in \Pi$.*

Theorem 11 *Unless $\mathbb{P} = \mathbb{NP}$, no pseudopolynomial algorithm can exist for NPC problem without numeric parameters.*

2.1.1 Algorithm for Partition

Here a pseudopolynomial algorithm for partition problem is presented. Input is the set $\{a_1, \ldots, a_n\}$. Let $B = \sum a_i/2$. The algorithm creates a matrix T, with elements defined thusly

$$t_{ij} = \begin{cases} 1 & \text{if } \exists \overline{N} \subseteq \{1, \ldots, n\} : \sum_{k \in \overline{N}} a_k = j \\ 0 & \text{otherwise} \end{cases}$$

The first row is very simple to create: for each a_i set $t_{0,a_i} = 1$. If row $i-1$ is already created, row i is created using these rules.

1. $\forall j : t_{i-1,j} = 1 \quad t_{ij} \leftarrow 1$. (If we could form sum S from $i-1$ first elements, this sum can be with equal success formed from first i elements.)
2. $\forall j : t_{i-1,j} = 1 \quad t_{i,j+a_i} \leftarrow 1$.

When the matrix is constructed, one should only look at $t_{n,B/2}$. Complexity is $O(nB/2)$.

2.1.2 Algorithm for Knapsack

You can devise the algorithm yourself.

2.1.3 Algorithm for Scheduling

Recall problem formulation. We have m processors, tasks $N = \{1, \ldots, n\}$ with execution times τ_i. The question is whether one can find a schedule with length less that T.

It turns out that if number of processors is fixed then a pseudopolymonial algorithm exists. Imagine a m-dimensional cube with side of T. Coordinate of each point inside cube indicated how many time for each processor is already allocated. It is obvious that if one more task is scheduled then one coordinate is increased and all the others are unaltered. It follows that points outside the cube shouldn't be considered, since they will result in no feasible schedule. In consequence, algorithm is very simple. Create a set of active points and let it to $\{(0, \ldots, 0)\}$. Then for each of n tasks, take all points and try every possible assignment of that task to processor, creating m new points for each one examined. Points that fall outside of cube should be rejected. If after processing all tasks you have at least one point inside cube, you're lucky. Complexity is $O(nmT^m)$.

2.1.4 Algorithm for Scheduling with Interrupts

It is known[6] that multiprocessor scheduling with interrupts has $O(n)$ complexity. Let's consider case when interrupts are not instant, but have time σ. It means that if task has stopped running on processor j_1 in time τ_1, moved to processor j_2, and started running there at time τ_2, then intervals $(\tau_1, \tau_1 + \sigma)$ for processor j_1 and $(\tau_2, \tau_2 + \sigma)$ for processor j_2 are used for interrupt handling.

[6] Actually from lectures on network algorithms

This problem is in $\mathbb{NPC}$. Reduction is, just like for scheduling without interrupts, from partition. Use theorem 9 on page 4 and let additionally $\sigma = 0$. In this case any feasible schedule will be interrupt-free and result of theorem 9 apply.

Pseudopolynomial algorithm exist, and is a modification of packing algorithm. The packing algorithm works rather simply – it places tasks to one processor until any time remains. It remaining time (τ) is less then task execution time (t), then only *last* time portion of task is executed on this processor, and all first part on some other, with interrupt between the tasks.

For our case, there's extra time for interrupt handling. If some task i is executed on two different processors, then the following inequality should hold $\exists t_i + 2\sigma \leqslant T$, otherwise no feasible schedule exists. Say that task runs from time 0 to time τ_2 on one processor, which then handles interrupt for time σ and yields task to another processor, which in turn handles interrupt for σ and the runs the task in interval (τ_1, T). It should be noted that $\tau_2 + \sigma \leqslant \tau_1 - \sigma$. Indeed, $\tau_1 = T - (t - \tau_2) = T - t + \tau_2 \geqslant 2\sigma + \tau_2$, from which we conclude that $\tau_2 + \sigma \leqslant \tau_1 - \sigma$.

Necessary condition of schedule existence is $\sum t_i \leqslant mT$. Compute $\alpha = \sum t_i + (m-1)2\sigma - mT$. If $\alpha \leqslant 0$ then schedule exists. Otherwise try to reduce interrupts count by $k_0 = \lceil \frac{\alpha}{2\sigma} \rceil$. In order to do it, try to split processors into k groups $k = k_0 + 1, \ldots, m$, so that tasks switches occur only within one group. To do it just treat every group as one processor with available time $(m_i T - (m_i - 1)2\sigma)$ and apply (a modification of) previous algorithm to assign n tasks to groups. If this is successful, further assignment within groups can always be done.

All we need is a method to enumerate all partitions[7]. We will generate partitions in reverse lexicographical order, which is defined thusly:

$$(m = m_1 + \cdots + m_k) \prec (m = a_1 + \cdots + a_p) \iff$$
$$\iff \exists t : m_i = a_i, \text{if } i < t \text{ and } m_t > a_t.$$

Additionally, numbers constituting partition will be sorted, therefore, no duplicate partition will be created. To get next partition these steps should be used.

1. $t = \max i : m_i > 1$. The partition looks like
   $m = m_1 + \cdots + m_{t-1} + \overbrace{m_t}^{>1} + \underbrace{1 + \cdots + 1}_{r \text{ times}}$
2. $s = m_t + r$
   $l = \lfloor \frac{s}{m_t - 1} \rfloor$
3. New partition is:
   $m = m_1 + \cdots + \underbrace{m_{t-1} + \cdots + m_{t-1}}_{l \text{ times}} + s \bmod (m_t - 1)$

Number of partitions if $O(\frac{1}{m} e^{\sqrt{m}})$, and every new partition costs $O(m)$ operation to create. So, if m is fixed, algorithm is pseudopolynomial.

2.2 Strong NP-completeness

Definition 3 *Π' is subproblem of Π if $\mathcal{D}_{\Pi'} \subseteq \mathcal{D}_\Pi$ and $Y_{\Pi'} = \mathcal{D}_{\Pi'} \cap Y_\Pi$.*

Definition 4 *Π_p is polynomial contraction of Π, if Π_p is subproblem Π and $\forall I \in \Pi_p \; M(I) \leqslant p(l(I))$.*

Definition 5 *Π is said to be strong NP-complete if $\Pi \in NP$ and $\exists p \Pi_p \in NPC$.*

Statement 1 *Unless $\mathbb{P} = \mathbb{NP}$, no strong NPC problem has a pseudopolynomial algorithm.*

[7] Mark the difference from meaning of "partition" used in other parts. Here we mean partition of a single number into a sum.

If such algorithm exists (with complexity $r(l(I), M(I))$), then $\forall p\, \Pi_p$ can be solved by this algorithm with complexity $O(r(l(I), p(l(I))))$, which means that NPC problem Π_p in also in P.

There are two methods to establish strong NP completeness.

1. Directly.
2. Using *pseudopolynomial reduction.*

Definition 6 (Pseudopolynomial reduction) $\Pi \leqslant_{pp} \Pi'$ *if* $\exists f : \mathcal{D}_\Pi \to \mathcal{D}_{\Pi'}$*:*

1. $\forall I \in \mathcal{D}_\Pi\ I \in Y_\Pi \iff f(I) \in Y_{\Pi'}$.
2. *f can be computed by a pseudopolynomial algorithm.*
3. $\exists p_1 : p_1(l'(f(I))) \geqslant l(I)$.
4. $\exists p_2 : M'(f(I)) \leqslant p_2(l(I), M(I))$.

Theorem 12 *If* Π *is strong NP,* $\Pi' \in \mathbb{NP}$ *and* $\Pi \leqslant_p \Pi'$*, then* Π' *is strong NP.*

Proof. Take arbitrary polynomial p. According to definition 5 on the page before $\Pi_p \in \mathbb{NPC}$. Then

$$\begin{array}{lll} (4) & \to & M'(f(I)) \leqslant p_2(l(I), M(I)) \leqslant \\ & \leqslant & p_2(l(I), p(l(I))) \leqslant \\ \langle 3 \rangle & \leqslant & p_2(p_1(l'(f(I))), p(p_1(l'(f(I))))) = \overline{p}(l'(f(I))) \end{array}$$

So, Π_p is reduced to polynomial reduction of Π'. Reduction is itself polynomial: its complexity is less than $r(l(I), M(I)) \leqslant r(l(I), p(l(I))) = \overline{r}(l(I))$. Therefore, $\Pi_{\overline{p}}$ is in $\mathbb{NPC}$, and Π is strong NPC.

Problem 24 (3-partitioning) *Given numbers* $a_1, \ldots, a_{3n}$ *and* B*, such that* $\sum a_i = nB$ *and* $\frac{B}{4} \leqslant a_i \leqslant \frac{B}{2}$*, is there partition* $N = N_1 \cup \ldots \cup N_n$ *(*$N_i \cap N_j = \emptyset$*) for which* $\forall j \sum_{i \in N_j} a_i = B$

This problem is strong NPC, but the proof is omitted.[8]

Theorem 13 *Ordering within interval is strong NPC.*

Proof. 3-partitioning $\leqslant_p p$ OWI(?). Reduction is similar to one used in proof of theorem 5: create $n-1$ auxiliary tasks and fix them evenly, making n intervals of length B each. And go on...

Theorem 14 *Multiprocessor scheduling without interrupts is strong NPC.*

Proof. Reduction is from 3-partitioning. Let $t_i = a_i$, $m = n$, $T = B$. Since bin packing is very similar to scheduling, the result apply to it.

Theorem 15 *Traveling salesman is strong NPC.*

Proof. Hamiltonian circuit is reducible to a pseudopolynomial contraction of TS. Just let

$$d_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A \\ 2 & \text{otherwise} \end{cases}$$

3 Optimization problems

Previously, only recognition problems were considered. In this section will be described notion of NP-equivalence, which indicates similar complexity as that of NPC problems.

[8] Omitted in lectures, not only in this text.

3.1 NP-hard, NP-easy and NP-equivalent problem

Definition 7 (Turing reducibility) $\Pi_1 \leqslant_t \Pi_2$ *iff* $\exists A_1$*, which solves* Π_1*, using algorithm* A_2 *for solving* Π_2*, and if* A_2 *– polynomial, then* A_1 *– polynomial too.*

Definition 8 (Class NPH) $\mathbb{NPH} = \{\Pi : \exists \Pi' \in \mathbb{NPC}, \Pi' \leqslant_t \Pi\}$

Every recognition problem is Turing-reducible to its optimization variant. Really, having optimal solution we can always compare the optimal result with boundary. Hence, all optimization variants of NPC recognition problems are in $\mathbb{NPH}$.

As an example of NPH problem consider *K-th set*.

Problem 25 (K-th set) *Given set* $\{a_1, \ldots, a_n\}$ *and numbers* B *and* K *(*$B \leqslant \sum a_i$*,* $K \leqslant 2^n$*), find if there's at least* K *subsets of* N *such that for each such subset* N' $\sum(a_i \in N') \leqslant K$*.*

Note: It is not known whether this problem is in $\mathbb{NP}$.

Theorem 16 *K-th set is in* $\mathbb{NPH}$*.*

Proof. Reduction from partition. For any instance of partition problem, do as prescribed:

1. Using binary search, find exact value of $K^* = |\{N' \subseteq N : \sum(a_i \in N') \leqslant \frac{B}{2}\}|$.
2. Invoke $\mathrm{KS}(\{a\}, \frac{B}{2} - 1, K^*)$. If the answer is "yes", it means that all sets with element sum $\leqslant \frac{B}{2}$ have element sum $\leqslant \frac{B}{2} - 1$ and, in consequence, $< \frac{B}{2}$. Thus, partition is not possible. If the answer is "no", then by the same considerations, partition exists.

Definition 9 (Class NPE) $NPE = \{\Pi : \exists \Pi' \in \mathbb{NPC}, \Pi \leqslant_t \Pi'\}$

Definition 10 (NP-equivalent problem) *Problem is said to be NP-equivalent if it is both in* $\mathbb{NPH}$ *and* $\mathbb{NPE}$*.*

3.2 Scheme of proving NP-equivalence

For each optimization problem two facts should be proved:

1. $\Pi \leqslant_t \Pi_{opt}$.
2. $\Pi_{opt} \leqslant_t \Pi$.

First fact is obvious. To prove the other we can create and auxiliary recognition problem $A\Pi$, which takes part of solution and tells if the solution can be augmented so that answer of Π is "yes". For traveling salesman problem the question of ATS is "is there a path passing through all the vertices with length less that so-and-so and starting with vertices such-and-such". Using auxiliary problem, we can try to prove two other facts:

1. $A\Pi \leqslant_t \Pi$.
2. $\Pi_{opt} \leqslant_t A\Pi$.

If both fact are proved, then $\Pi_{opt} \leqslant_t \Pi$, which is what we want. If $A\Pi \in \mathbb{NP}$, the first fact follows directly from definition of $\mathbb{NP}$. The second fast should be proved explicitly.

Now the proof is presented for traveling salesman problem. Say that ATS exists and accepts parameters n_i, d_{ij}, B and partial path $\pi = \{i_1, \ldots, i_k\}$. Solving the TS_{opt} consist of two parts:

1. Finding the minimum path length. It is bounded by $n * \max(d_{ij})$. Using binary search minimum path length can be found with at most $\lceil \log_2[n * \max(d_{ij})] \rceil$ invocations of ATS.
2. Finding the path. Let l^* denote the minimum length found above. Then, proceed as specified below

(a) Let $\pi = \{\}$.

(b) $\forall i \in \overline{1,n} : i \notin \pi$ invoke $ATS(n, d_{ij}, l^*, (\pi|i))$.[9] For some i^* the call will return true. Append i^* to π.

(c) Repeat the above step n times.

This part will invoke ATS $O(n^2)$ times.

We conclude that reduction requires polynomial number of invocations of ATS, and $ATS \leqslant_t TS_{opt}$.

4 Methods of solving NPC problems

4.1 Approximate algorithms

Consider a generic optimization problem: we have set $\mathcal{D}_\Pi$ and for each $I \in \mathcal{D}_\Pi$ have to minimize some function f over set of all feasible solutions $X(I)$, i.e. $?x^* : min_{x \in X(I)} = f(x^*(I))$. Approximate algorithms in general don't return x^* but some value x_A, which should be "close" to x^*. For an instance of a problem, these notions of "closeness" can be used:

1. $r_a^1 = \frac{f(x_A(I))}{f(x^*(I))}$
2. $r_a^2 = f(x_A(I)) - f(x^*(I))$
3. $r_a^3 = \frac{f(x_A(I)) - f(x^*(I))}{f(x^*(I))}$

For approximate algorithm similar metrics exist

$$r_A^1 = \sup_{I \in \mathcal{D}_\Pi} r_A^1(I) \quad r_A^2 = \sup_{I \in \mathcal{D}_\Pi} r_A^2(I) \quad r_A^2 = \sup_{I \in \mathcal{D}_\Pi} r_A^3(I)$$

4.1.1 Bin packing

Consider a naive approach. For each object, pick the first bin that have enough volume to hold the object.[10]

At most one bin can be filled with half of capacity or less. Really, is bins j_1 and j_2 ($j_1 < j_2$) are both filled with $V/2$ volume, then all content of j_2 can be moved j_1, and since the algorithm always considers j_1 before j_2, it won't fail to do it. It follows that, at worst, two times more bins will be used. Formally: $\sum v_i > \frac{v}{2} f(x_A(I)) \Rightarrow f(x_A(I) < 2(\frac{\sum v_i}{V}) \leqslant 2f(x^*(I))$ So, for this algorithm $r_A^1 \leqslant 2$.

4.1.2 Scheduling

Another greedy algorithm with similar quality comes here. We have m processors and n tasks with execution times t_i. Take each task at a moment, and compute $L_1, \ldots, L_m$ – time that is already allocated on each processor. Schedule the task to the processor with minimum L_i.

Assume that after the algorithm finished $L_{j_1} = max_{j=\overline{1,n}} L_j$ and that last task scheduled to processor j_1 is task i. Let L'_{j_1} denote time that was allocated to processor j_1 just before task i was scheduled. These inequalities hold:

1. $t_i \leqslant f(x^*(I))$. Obvious.
2. $L'_{j_1} \leqslant f(x^*(I))$. Just before task i was scheduled, L'_{j_1} was the smallest time already allocated. It means that if no task after i is scheduled at all, and all previous tasks are allocated super-optimally, with equal time used on every processor[11], then total time will not be less than L'_{j_1}, just because arithmetic mean is not less that the minimum element.

[9] vertical line means list concatenation

[10] Such algorithms are often called "greedy"

[11] Such scheduling is, in general, not possible, that why I call it super-optimal

Net result is that $L_{j_1} \leqslant 2f(x^*(I))$ and $r_A^{'} \leqslant 2$. For every individual problem $f(x^*(I)) \geqslant (\frac{\sum t_i}{m}) \stackrel{df}{=} A$, and $r_A^1 = \frac{f(x_A(I))}{f(x^*(I))} \leqslant \frac{f(x_A(I))}{A}$.

4.1.3 Traveling salesman with triangle inequality

Further discussion assumes that lengths in traveling salesman problem satisfy triangle inequality and that every pair of towns is connected.

We start with finding *shortest spanning tree*(SST). Spanning tree is a tree which contains every vertex of a graph. Shortest spanning tree has the minimum sum of edge lengths. For the case of an undirected graph, SST can be found as described below:

1. $SST \leftarrow (i_1)$
2. $(Assumethat)\ SST = (V^{'}, A^{'})$
   $?(u, v) : \min_{(i,j)\in A, i\in V^{'}, j\in V\setminus V^{'}} d_{ij} = d_{uv}$
   $SST \leftarrow (v^{'} \cup v, A^{'} \cup (u, v))$

Note: The algorithm won't work for directed graph – in particular, because spanning tree in directed graph can't have any vertex as root. It imposes additional constraint that $d_{ij} = d_{ji}$

Statement 2 *At each step, we have tree, which is part of some SST.*

Proof. Induction on tree size.

1. Tree of 1 vertex – obvious.
2. Say we have tree T_k, which is part of some SST. Assume that arc (u, v) that we gonna append to T_k isn't part of the SST. We will prove that it it part of some other SST.
   If $(u, v) \notin SST$, then there's path $(u, \ldots, x, y, \ldots, v)$, where $x \in T_k$ and $y \in SST \setminus T_k$. It follows from the fact that in any undirected tree there's path between any two vertices, and also because $u \in T_k$ and $v \notin T_k$. We know that $d_{uv} \leqslant d_{xy}$. So, by removing edge (x, y) and adding (u, v), we get new tree, which is also SST.

Path of salesman can be found thusly.

1. Find $SST(G)$.
2. Traverse the SST, returning to where traversal started. Each edge will be traversed two times.
3. Improve the path. Move the same way as previously, but this time skip already traversed vertices: find next unvisited vertex and go there directly. It is possible since the graph is complete, and path length won't increase thanks to triangle inequality.

Result estimation

1. $L_{SST} \leqslant L^*$. Exact result is circuit, which is tree plus one extra edge. Its length L^* can't be less then length of SST.
2. $L < 2 * L_{SST}$.
3. So, $L < 2 * L^*$ and $r_a^1 < 2$.

Another algorithm used Euler's circuits – circuits that pass every edge in a graph exactly once. Such circuit exist in a graph iff degree of each vertex is even. Such graphs are called Euler's. [Proof is omitted]

Algorithm is

1. Find SST in graph G.

2. Make Euler's graph of SST.
   $V' = \{v \in V : degree(v) \bmod 2 = 1\}$
   $|V'|$ – even, because $\sum_{v \in V} degree(v)$ is even for every graph.
   $G' = (V', A')$
   Find minimum pair-matching in G' (pair-matching with minimum sum of edge weight), using, for example, min-cost flow algorithm. Append edges from pair-matching to SST.

3. SST with added edges is Euler's graph. Find Euler's circuit in it. Build salesman path from it by skipping over already visited vertices.

Result estimation

1. Resulting path is not longer than $L_{SST} + U$, where U is sum of lengths of edges in pair-matching. Let's prove that $U \leqslant 0.5L^*$. Take optimal salesman path. Skip every vertex not in V'. Each remaining edge joins two vertices from V'. Assign numbers to edges and two sets E_1 and E_2: of edges with odd numbers, and of edged with even number. Both set correspond to some pair matching so: $U \leqslant \min(U_1 = \sum_{(u,v) \in E_1} d_{ij}, U_2 = \sum_{(u,v) \in E_2} d_{ij})$
   Since also $U_1 + U_2 \leqslant L^*$, $U \leqslant \frac{L^*}{2}$.

4.1.4 Knapsack

Previously, in section 2.1.2 on page 7, we hinted on how pseudopolynomial algorithm for knapsack problem can be constructed. Using similar approach, approximate algorithm can be build. Let $S = \max s_i$ and create matrix $A \in \{true, false\}^{n,nS}$ with elements

$$a(i,s) = \min(0,v) : \exists(\overline{N} \subseteq \{1,\ldots,n\} : \sum_{i \in \overline{N}} s_i = S \ \sum_{i \in \overline{N}} v_i = v \leqslant V)$$

This matrix can be filled as usual, with complexity $O(n^2 S)$.

Select a number t and modify all s_i by setting t lowest bits to zero. We'll denote the new values as $s_i^{'}$. If N_1 and N_2 are optimal sets for modified and original problems, correspondingly, the following holds:

$$r_A^3 = \frac{\sum_{i \in N_1} s_i - \sum_{i \in N_2} s_i}{\sum_{i \in N_2} s_i} \leqslant \frac{\sum_{i \in N_1} s_i - \sum_{i \in N_2} s_i^{'}}{\sum_{i \in N_2} s_i} \leqslant \frac{\sum_{i \in N_1} s_i - \sum_{i \in N_1} s_i^{'}}{\sum_{i \in N_2} s_i} \leqslant \frac{n10^t}{S}$$

If r_A^3 is to be limited by ϵ, equality $\frac{n10^t}{S} = \epsilon$ is the starting point. From it we deduce that $t = \lfloor \lg \frac{\epsilon S}{n} \rfloor$, and that complexity should be computed using $\frac{S}{10^t}$ instead of S, and will be equal to $O(n^2 \frac{S}{10^t}) = O(n^3/\epsilon)$.

4.1.5 Vertex cover

Algorithm:

1. $V' \leftarrow \emptyset$, $A' \leftarrow A$

2. Pick any $(u,v) \in A'$. $V' += \{u,v\}$. Remove (u,v) and all edges incident either to u or v from A'.

3. Repeat until $A' = \emptyset$.

If k is the number of picked edges, then $|V'| = 2k$. Also, by construction, picked edges share no endpoint, so $|V^*| \geqslant k$. We conclude that $r_A^1 \leqslant 2$.

4.1.6 Negative results

Statement 3 *Unless $P = \mathbb{NP} \nexists$ approximate polynomial algorithm for solving traveling salesman problem with $r_A^1 \leqslant K$, for any constant K.*

Proof. If such algorithm exist, for every individual problem construct another one thusly:

$$G = (V, A) \rightarrow n = |V|, d_{ij} = \begin{cases} 1 & (i,j) \in A \\ nK & \text{otherwise} \end{cases}$$

Solution of this problem with $r_A^1 \leqslant K$ can be converted into exact solution of the original problem.

Statement 4 *Unless $\mathbb{P} = \mathbb{NP} \nexists$ approximate polynomial algorithm for knapsack problem with $r_A^2 \leqslant K$, for any constant K.*

Proof. Assume contrary. Then for every I create another instance $N' = N_1$, $v_i^{'} = v_i$, $s_i^{'} = (K+1)s_i$. Optimal sets are the same. However, if new problem can be solved with $r_A^1 \leqslant K$, we get $f(x^*(I)) - f(x_A(I)) \leqslant K$, but left part of this inequality is divisible by $k+1$. Hence, the difference is equal to zero, and exact solution of the modified problem, as well as original one, is computed.

Statement 5 *Unless $\mathbb{P} = \mathbb{NP} \nexists$ approximate polynomial algorithm for vertex cover problem with $r_A^2 \leqslant K$, for any constant K.*

Proof. Make $K+1$ copies of original graph and apply the algorithm to the resulting graph. In at least one copy, vertex cover will be optimal. Otherwise, total number of vertices in vertex cover will exceed optimal by at least $K+1$.

4.2 Search algorithms

Search algorithms, otherwise known as "branch-and-bound" algorithms, address the complexity of NPC problems by pruning the search tree using various heuristic considerations.

Suppose that the problem formulation is $min_{x \in X} f(x)$ and that X is finite. Take $X_1, X_2 : X_1 \cap X_2 = X, X_1 \cup X_2 = \emptyset$. If we can establish without much computational complexity that

$$F_{L1} \leqslant \min_{x \in X_1} f(x) \leqslant F_{U1}$$
$$F_{L2} \leqslant \min_{x \in X_2} f(x) \leqslant F_{U2}$$

and it happens that $F_{L2} > F_{U1}$, then X_2 doesn't contain the solution and shouldn't be searched. Branch-and-bound algorithms consists, thusly, of two parts – partitioning of the search space and evaluation/rejection.

4.2.1 Shortest path

Partition: According to starting edges in the path.

Evaluation: If l is total length of initial part, then $F_L = l$ and $F_H = l + F$, where F is shortest remaining path as computed by some fast algorithm, e.g. greedy one.

4.2.2 Scheduling

Partition: Somehow.

Evaluation: Somehow.

4.3 Randomized algorithms

Some facts from probability theory:

I. Conditional probability of event A on condition that event B occurred is $P(A|B) = \frac{P(AB)}{P(B)}$. It is defined only if $P(B) \neq 0$.

II. Total probability. Let A, $B_1, \ldots, B_n$ be events. If $P(B_iB_j) = 0, i \neq j$ and $P(\cup_i B_i) = 1$, then $P(A) = P(A|B_1)P(B_1) + \ldots + P(A|B_n)P(B_n)$.

4.3.1 Polynomials equivalence

Consider a problem: find is $g(x_1, \ldots, x_n) \stackrel{?}{\equiv} h(x_1, \ldots, x_n)$ Before presenting randomized algorithm for the problem, we need one fact.

Lemma 1 (Shwartz's lemma) *If $f(x_1, \ldots, x_n)$ with degree of any variable $\leqslant k$, $f \not\equiv 0$, $\xi_1, \ldots, \xi_n$ – independent random variables uniformly distributed over integer values in range $[0, N-1]$, for some N then $P(f(\xi_1, \ldots, \xi_n) = 0) \leqslant \frac{kn}{N}$*

Proof. Use induction on the number of variables.

a) $n = 1$. Polynomial has no more than k roots. If all those roots are integers in the range $[0, N-1]$, $P(f(\xi_1) = 0) = \frac{k}{N}$. If roots are not such luckily placed, probability will be even less.

b) $n - 1 \to n$. Use this representation $f(x_1, \ldots, x_n) = f_0(x_2, \ldots, x_n) + f_1(x_1, \ldots, x_n)x_1 + \ldots + f_t(x_2, \ldots, x_n)x_1^t$, $t \leqslant k$. Note that $f_t(\xi_2, \ldots, \xi_n) \neq 0$. Using total probability formulae, we can write

$$\begin{aligned} P(f(\xi_1, \ldots, \xi_n)) &= \overbrace{P(f()|f_t(\xi_2, \ldots, \xi_n) = 0)}^{\leqslant 1} P(f_t(\xi_2, \ldots, \xi_n) = 0) + \\ &+ P(f()|f_t(\xi_2, \ldots, \xi_n) \neq 0) \overbrace{P(f_t(\xi_2, \ldots, \xi_n) \neq 0)}^{\leqslant 1} \leqslant \\ &\leqslant P(f_t(\xi_2, \ldots, \xi_n) = 0) + P(f(\xi_1, \ldots, \xi_n) = 0|f_t(\xi_2, \ldots, \xi_n) \neq 0) \leqslant \\ &\leqslant \frac{k(n-1)}{N} + \frac{k}{N} = \frac{kn}{N} \end{aligned}$$

We are ready to describe the algorithm for polynomials equivalence checking. The question can be formulated as $f \stackrel{?}{\equiv} 0$. Assume that function f can be effectively computed. Generate random integer values uniformly distributed in range $[0, 2kn-1]$ and compute the value of the function in that random point. If it is not zero, result is computed. Otherwise make another try, etc. If $f \not\equiv 0$, then, according to lemma, $P(\bar{\xi}) = 0) \leqslant \frac{kn}{2kn} = \frac{1}{2}$. Since all random values are independent, probability to get M zeroes when $f \not\equiv 0$ is less than $(\frac{1}{2})^M$. With sufficiently large M, $f \equiv 0$ is very probable.

4.3.2 Pair-matching

Pair-matching can be solved using the previous algorithm by means of a simple auxiliary construction. Create matrix M:

$$m_{ij} = \begin{cases} x_{ij} & \text{if } (a_i, b_j) \in A \\ 0 & \text{otherwise} \end{cases}$$

Statement 6 *Full pair-matching exists iff $det(M) \not\equiv 0$.*

Proof. If $det(M) \equiv 0$ it means that every $(\ldots)$ contains zero. In fact, if any $()$ has no zero, then by increasing at the same time every variable in $()$, we can make absolute value of determinant arbitrary large. If every $()$ contains zero, then pair-matching is not possible, since for every possible correspondence, presence of zero means that some pair can't be joined because of absence of an edge.

Determinant is a polynomial. One can compute if effectively using Gaussian elimination. So, previous algorithm can be applied in this case.

5 Additional proofs

Here we will establish NP-completeness for tasks that were not covered in the lecture course.

Currently, proofs can be only told in a personal communication.
(If you have any interest in the topic.)

5.1 3DM

5.2 Partition

5.3 Graph coloring

5.4 Hamiltonian circuit