Chapter 1

Abstract Complexity

1.1 Introduction

Time complexity: $T(n)$ where n: input size.

$$T(n) = \max\{\#\text{steps for input } x \mid |x| = n\}$$

Definition 1.1.1. If there exists a polynomial p such that: $\forall n, T(n) \leq p(n)$ ($T(n) = O(poly)$) then the algorithm is *efficient* (*Edmonds 1968*).

1.2 Optimization Problems

For each instance x there is a set of *feasible* solutions $F(x)$. To each $s \in F(x)$, we assign, using an objective function c, a positive integer $c(s)$. We search for the solution $s \in F(x)$, for which $c(s)$ is minimum (or maximum).

Example: *Traveling Salesperson Problem*: Given a finite set $C = \{c_1, c_2, \ldots, c_n\}$ of cities and a distance $d(c_i, c_j) \in Z^+, \forall (c_i, c_j) \in C^2$. We ask for a *tour* that passes exactly one time through each city and comes back to the original city, with minimum total distance.

1.3 Decision Problems

Decision problems have answers of the form: 'yes' or 'no'. For each optimization problem, we can create a corresponding decision problem:

Example 1.3.1. 'Given graph $G(V, E)$. What is the maximum size clique in G?': an optimization problem.

The corresponding decision problem: 'Given graph $G(V, E)$ and a number k. Is there a clique with k (or more) nodes in G?'

1.4 Decision problems and languages

One of the reasons we consider decision problems is that they are related to *formal languages*.

For an encoding e of the input, using the alphabet Σ, we associate the following language with the decision problem Π:

$$L(\Pi) = \{x \in \Sigma^* \mid x \text{ is a representation of a 'yes' instance of problem } \Pi\ \},$$

when encoding e is used.

1.5 Classes P and NP

P (*Polynomial*): class of problems solvable in polynomial time by some deterministic algorithm.

$$P = \{L \mid \exists \text{ polynomial time DTM that decides language } L\}$$

NP (*Non-deterministic Polynomial Time*): class of problems solvable in polynomial time by some non deterministic algorithm.

Example 1.5.1. Searching in a non sorted array $a[n]$. Deterministically, time $\Theta(n)$ is needed. Non deterministically, we just need $\Theta(1)$ time:

```
choose a[i]
verify : if a[i]=x then found
```

Example 1.5.2. Sorting n elements. Deterministically we need $\Theta(n \log n)$ time. Non deterministically, time $\Theta(n)$ is needed:

```
choose a permutation
verify : a[i]<a[i+1] for all i
```

In general, for the hard problems in NP, finding a solution is slow, but checking the correctness of a proposed solution is fast.

Example 1.5.3. SAT: Given variables x_i taking the values True or False. We define:

- literals: terms $x_i, \neg x_i$,
- clauses: disjunctions of literals $literal_1 \vee literal_2 \vee \ldots \vee literal_m$

- CNF (Conjunctive Normal Form):

$$clause_1 \wedge clause_2 \wedge \ldots \wedge clause_n$$

The *Satisfiability* problem:

SAT
Given: A boolean formula in CNF.
Question: Is there an assignment that satisfies the formula (i.e., the formula evaluates to True)?

Deterministically, we need $O(2^n)$ time (exponential). Non deterministically, we can solve the problem in polynomial time:

- Choose a truth assignment.
- Check if the assignment satisfies the boolean formula.

A NDTM accepts, if there is at least one accepting computation path. The time complexity of a NDTM program is defined:

$$T(n) = \begin{cases} \max\limits_{|x|=n}\{\text{min \# steps for accepting } x\}, & \text{if } x \text{ acceptable} \\ 1, & \text{otherwise} \end{cases}$$

A NDTM program M is of polynomial time if:

$$\exists \text{ polynomial } p: \ T(n) \leq p(n), \forall n$$

$$NP = \{L \mid \exists \text{ a polynomial time NDTM that decides language } L\}$$

1.5.1 Relation between P and NP

Obviously: $P \subseteq NP$.

A non deterministic polynomial time algorithm can be simulated by a deterministic one using exponential time: $NP \subseteq DEXPTIME$.

Most researchers believe that P is different from NP, but this is still an open problem.

1.6 Reductions

Definition 1.6.1 (Karp reduction or transformation)**.**

$$A \leq_m^P B\colon \quad \exists f \in \mathrm{FP}, \forall x(x \in A \iff f(x) \in B)$$

Definition 1.6.2 (Log-space reduction)**.**

$$A \leq_m^L B\colon \quad \exists f \in \mathrm{FL}, \forall x(x \in A \iff f(x) \in B)$$

We have $A \leq_m^L B \implies A \leq_m^P B$, but not the converse.

Definition 1.6.3. Class C is *closed* under reduction $\leq$ if

$$A \leq B \wedge B \in C \implies A \in C.$$

Definition 1.6.4 (Hardness)**.** A is C-hard, under $\leq$, if:

$$\forall B \in C : B \leq A.$$

Definition 1.6.5 (Completeness)**.** A is C-complete, under $\leq$, if:

$$A \text{ is } C - \text{hard under} \leq \quad \wedge \quad A \in C.$$

So a problem L is **NP-complete** under $\leq_m^p$ if:

$$(L \in NP) \wedge (\forall L' \in NP :\ L' \leq_m^p L)$$

NP-complete problems are the hardest of class NP. If an **NP-complete** problem is proved to be in P, then all problems in NP are in P, i.e., P = NP.

Lemma 1.6.6. If $L_1 \leq_m^p L_2$, L_1 is **NP-complete** and $L_2 \in NP$ then L_2 is **NP-complete**.

The lemma indicates the way we use reduction; we try to reduce a known NP-complete problem to another problem in NP, thus showing that the latter problem is P-complete too. In order to do that, we need to know some NP-complete problem to start from, and then reduce it to other NP problems. This 'first' NP-complete problem was given by Cook with his theorem.

1.6.1 Cook's Theorem

SAT problem
Given: A boolean formula in CNF.
Question: Is the boolean frmula satisfiable?

Example 1.6.7.

- Formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ is satisfiable. A satisying truth assignment is $(x_1, x_2) = (T, T)$.
- Formula $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge \neg x_1$ is not satisfiable.

Theorem 1.6.8 (Cook)**.** *The SAT problem is NP-complete.*

Proof. Obviously SAT is in NP: Choose non-deterministically a satisfying assignment and check (in polynomial time) if the formula evaluates to true. Next, we must prove that all problems in NP can be reduced to SAT. In fact, we prove: that any computation of a polynomial time NDTM M can be represented by a boolean formula Φ of polynomial length, such that:

> There is a computation of M that accepts x iff there is a satisfying assignment for boolean formula $\Phi(x)$.

Initially, the head of the NDTM is at position 0 and only positions from $-p(n)$ to $p(n)$ can be written because M is polynomial time. Computation ends after $p(n)$ steps.

Possible states of the TM:

$$Q = \{q_0, q_1, \ldots, q_r\}, \qquad q_1 = q_Y, \quad q_2 = q_N$$

Possible symbols of the TM:

$$\Gamma = \{s_0, s_1, \ldots, s_v\}, \qquad s_0 = \text{blank}$$

Boolean variables of the formula Φ:

- $Q[i,k],\ 0 \le i \le p(n),\ 0 \le k \le r$: At step i (point of time i) TM is in state q_k. $((r+1) \cdot (p(n)+1)$ many).
- $H[i,j],\ 0 \le i \le p(n),\ -p(n) \le j \le p(n)$ At step i the head is at position j. $((p(n)+1) \cdot (2p(n)+1)$ many).
- $S[i,j,l],\ 0 \le i \le p(n),\ -p(n) \le j \le p(n),\ 0 \le l \le v$: The symbol at position j, at step i, is s_l. $((p(n)+1) \cdot (2p(n)+1) \cdot (v+1)$ many).

Number of variables: $O(p^2(n))$ (because r, v are constants for a specific NDTM), i.e., polynomial w.r.t. n.

Clauses of formula $\Phi(x)$ are divided in 6 groups:

- G_1: For all i, at step i, the TM is at exactly one state:

$$\wedge_{i=0}^{p(n)}((\vee_{j=0}^{r}Q[i,j])\wedge(\wedge_{j=0}^{r}\wedge_{k=0}^{j-1}(\neg Q[i,j]\vee\neg Q[i,k]))) \tag{1.1}$$

  Total number of literals in those clauses:

$$(p(n)+1)\cdot[(r+1)+\frac{r(r+1)}{2}\cdot 2]=(p(n)+1)\cdot(r+1)^2=O(p(n))$$

- G_2: At step i, the TM's head is at exactly one position:

$$\wedge_{i=0}^{p(n)}((\vee_{j=-p(n)}^{p(n)}H[i,j])\wedge(\wedge_{j=-p(n)}^{p(n)}\wedge_{k=-p(n)}^{j-1}(\neg H[i,j]\vee\neg H[i,k]))) \tag{1.2}$$

  Total number of literals in those clauses:

$$(p(n)+1)\cdot(2p(n)+1)^2)=O(p^3(n))$$

- G_3: At step i, every position in the tape, contains exactly one symbol:

$$\wedge_{i=0}^{p(n)}\wedge_{j=-p(n)}^{p(n)}((\vee_{l=0}^{v}S[i,j,l])\wedge(\wedge_{l=0}^{v}\wedge_{k=0}^{l-1}(\neg S[i,j,l]\vee\neg S[i,j,k]))) \tag{1.3}$$

  Total

$$(p(n)+1)\cdot(2p(n)+1)\cdot(v+1)^2=O(p^2(n))$$

- G_4: At step 0, the TM is at the initial configuration: state is q_0: $Q[0,0]$, head at position 1: $H[0,1]$, in positions 1 to n of the tape is the input x: $S[0,1,l_1]\wedge S[0,2,l_2]\wedge\ldots\wedge S[0,n,l_n]$, for $x=S_{l_1},S_{l_2},\ldots S_{l_n}$, all other positions are blank. Total number of literals: $2p(n)+3=O(p(n))$.

- G_5: This group has just one literal that says that the computation is accepting: the state is $q_1=q_Y$ at the end of the computation, i.e.:

$$Q[p(n),1]$$

- G_6: (a) The content of the tape at position j, cannot be changed at step $i+1$, if at step i the head was not at position j:

$$(\neg H[i,j]\wedge S[i,j,l])\rightarrow S[i+1,j,l],$$

  or equivalently:

$$\begin{cases}(H[i,j]\vee\neg S[i,j,l])\vee S[i+1,j,l]\\ 0\leq i\leq p(n),-p(n)\leq j\leq p(n),0\leq l\leq v\end{cases}$$

(b) The TM's configuration at step $i+1$ is derived from applying the the TM's transition function δ on the configuration at step i:

$$(H[i,j] \wedge Q[i,k] \wedge S[i,j,l]) \rightarrow \\ \vee_{m=1}^{|\delta(q_k,s_l)|} (H[i+1, j+\Delta_m] \wedge Q[i+1,k'_m] \wedge S[i+1,j,l'_m]) \quad (1.4)$$

$$(q_{k'_m}, s_{l'_m}, \Delta_m) \in \delta(q_k, s_l), \text{ where} \Delta_m \in \{-1, 0, 1\}.$$

Total number of literals in G_6: $O(p^2(n))$.

The final formula is

$$\Phi(x) = G_1 \wedge G_2 \wedge G_3 \wedge G_4 \wedge G_5 \wedge G_6$$

Length: $O(p^3(n))$

If there is a computation of the NDTM M that accepts x in time $p(n)$, then $\Phi(x)$ is satisfiable, by construction. Conversely, if $\Phi(x)$ is satisfiable, then this satisfying truth assignment corresponds to a computation of M that accepts x. **Thus SAT is NP-complete**. □

Definition 1.6.9 (Cook reduction)**.**

$$A \leq_T^P B \colon A \in P^B$$

i.e A can be computed in polynomial time by a DTM that uses an oracle for B, by putting strings on an extra tape, called the query tape, and obtaining membership answer for set B by the oracle, an outside agent who can answers, correctly and for free, any questions about membership to B.

Facts:

$$A \leq_m^P B \quad \Rightarrow \quad A \leq_T^P B$$

$$B \text{ is NP-Complete w.r.t } \leq_m^P \quad \Rightarrow \quad B \text{ is NP-Complete w.r.t } \leq_T^P$$

Chapter 2

Transformations of problems

2.1 Introduction

Sometimes we transform the original problem to another problem, for which we can easily find the solution. Afterwards we transfer the solution to the original problem.

Example

Given: A chessboard 3×3 with black (X) and white(O) knights as you see in figure 2.1.

Question: How can we swap the positions of the white and the black knights using legal moves?

X		X
O		O

Figure 2.1: Knight Problem on Chessboard

Solution: It is enough to number positions on the chessboard based on the possibility of moves, see figure 2.2.

1	6	3
4		8
7	2	5

Figure 2.2: Numbering of chessboard 3×3

We build a ring as we see in figure 2.3. The black knights are in position 1, 3 and the white in position 5, 7. The solution is now obvious.

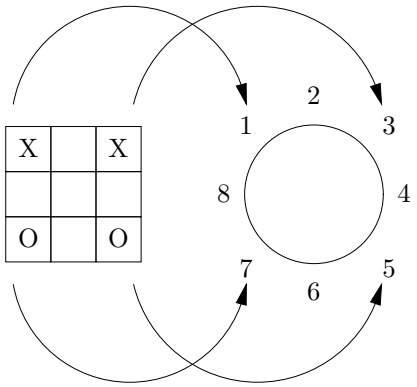

Figure 2.3: Transformation of the knight problem

Here we use the reduction method in order to show a problem is difficult to solve: We reduce with an easy transformation, a known hard problem to our new problem and that shows that it is also hard.

In order to show that problem Π is NP-complete we follow the steps:

1. show $\Pi \in NP$.

2. Choose a known NP-complete problem Π' and constructing a function f, transform it to problem Π.

3. Show that the transformation f can be done in polynomial time

4. Prove that: $x \in \Pi' \iff f(x) \in \Pi$.

We will demonstrate the reductions of problems as shown in figure 2.4. Historically, most of them were presented by *Karp* (1972).

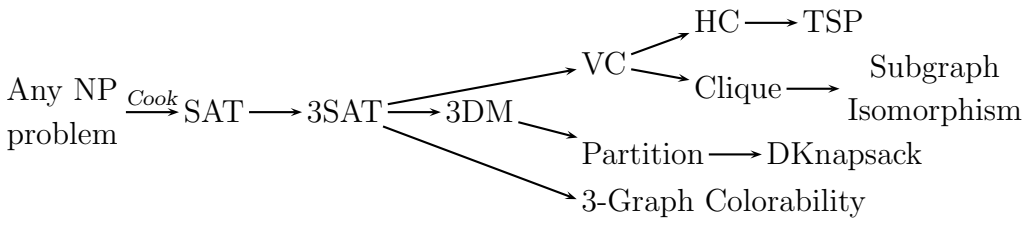

Figure 2.4: Reductions of problems in NP

2.2 Definitions of Decision Problems

SAT (SATISFIABILITY)
Given: A boolean formula in CNF.
Question: Is the boolean expression satisfiable? I.e., does there exist a truth assignment so that the boolean formula obtains the value True.

3SAT
Given: A boolean formula in CNF, so that every clause has exactly 3 literals.
Question: Is the boolean formula satisfiable?

VC (VERTEX COVER)
Given: A graph $G(V, E)$ and a positive integer $k \leq |V|$.
Question: Does there exist a vertex cover of all the edges of E, of size $\leq k$; i.e., does there exist a set $V' \subseteq V$ such that $|V'| \leq k$ and $\forall \{u, v\} \in E : \ u \in V' \vee v \in V'$;

3DM (3-DIMENSIONAL MATCHING)
Given: A set $M \subseteq W \times X \times Y$, were W, X, Y are disjoint sets with $|W| = |X| = |Y| = q$.
Question: Does M contain a matching; i.e., Does there exist a set $M' \subseteq M$ such that $|M'| = q$ and any 2 elements of M' have no common coordinate?

GRAPH 3-COLORABILITY
Given: Given a graph $G(V, E)$.
Question: Can we color the nodes of the graph G using **3 colors** so that any adjacent nodes have different colors? I.e., Does there exist a function $f : \ V \to \{0, 1, 2\}$ such that, $\forall (u, v) \in E : f(u) \neq f(v)$?

HC (Hamilton Circuit)
Given: A graph $G(V, E)$.
Question: Does the graph have a Hamilton Cycle? I.e., does there exist a permutation of all nodes of graph G, $\langle v_1, v_2, \ldots, v_n \rangle, n = |V|$, such that

$$(v_i, v_{i+1}) \in E, 1 \leq i \leq n-1, \text{ and } (v_n, v_1) \in E;$$

TSP (TRAVELING SALESMAN PROBLEM)
Given: A complete graph $G(V, E)$ with weights and a number B.
Question: Does there exist a tour that goes through all the nodes of G, $< v_{\pi_{(1)}}, v_{\pi_{(2)}}, \ldots, v_{\pi_{(m)}} >$ such that:

$$\sum w(v_{\pi_{(i)}}, v_{\pi_{(i+1)}}) \ + w(v_{\pi_{(m)}}, v_{\pi_{(1)}}) \leq B;$$

CLIQUE
Given: A graph $G(V, E)$ and a positive integer $j \leq |V|$.
Question: Does the graph G contain a clique of size $\geq j$? I.e., Does there exist $V' \subseteq V$, such that: $|V'| \geq j$ and $\forall u, v \in V' : (u, v) \in E$; Or equivalently: Does the graph G contain a complete subgraph with number of nodes $\geq j$;

SUBGRAPH ISOMORPHISM
Given: Two graphs $G(V_1, E_1)$ and $H(V_2, E_2)$.
Question: Does graph G contain a subgraph which is isomorphic to H? I.e., do there exist $V \subseteq V_1, E \subseteq E_1$ such that $|V| = |V_2|, |E| = |E_2|$ and a function $f : V_2 \to V$, 1-1 and onto (bijection) such that: $(u, v) \in E_2 \iff (f(u), f(v)) \in E$;

PARTITION
Given: A finite set A with weights, $w(a) \in Z^+, \forall a \in A$.
Question: Is it possible to split the set A into two subsets of equal total weight ? I.e., does there exist $A' \subseteq A$ such that,

$$\sum_{a \in A'} w(a) = \sum_{a \in (A - A')} w(a);$$

DKNAPSACK (DISCRETE KNAPSACK)
Given: A finite set U, with a weight function $w(u) \in Z^+, \forall u \in U$, a cost function $p(u) \in Z^+, \forall u \in U$ and two positive integers W, P.
Question: Can we take some objects from set U and put them in the knapsack so that the total weight of the knapsack $\leq W$ and the total value $\geq P$? I.e., does there exist a $U' \subseteq U$ such that,

$$\sum_{u \in U'} w(u) \leq W \text{ and } \sum_{u \in U'} p(u) \geq P;$$

2.3 Reduction of SAT to 3SAT

Theorem 2.3.1. *3SAT is* **NP-complete**.

Proof. It is easy to see 3SAT $\in NP$. A nondeterministic algorithm can guess a truth assignment and then verify in polynomial time if it satisfies the boolean formula.

To show that 3SAT is NP-complete, we will reduce SAT to it (SAT $\leq_m^p$ 3SAT). Assume that an instance of SAT is given, i.e a set C of m clauses, $C = \{c_1, c_2, \ldots, c_m\}$ that use variables from a set of n variables, $U = \{z_1, z_2, \ldots, z_n\}$. We construct a new set of clauses C' and a new set of new variables V', so that every clause in C' has exactly 3 literals.

- For every clause $c \in C$ of the original formula that consists of 1 literal $c = z$, construct the following 4 clauses (Variables y_1 and y_2 are new , not in C):

  $$(z \vee y_1 \vee y_2) \wedge (z \vee y_1 \vee \neg y_2) \wedge (z \vee \neg y_1 \vee y_2) \wedge (z \vee \neg y_1 \vee \neg y_2)$$

  It is easy to see that the value of this expression is always the same as the value of z, independent of the values of y_1, y_2 (*dummy variables*).

- For every clause $c \in C$ of the original formula that consists of 2 literals $c = z_1 \vee z_2$, construct the following 2 clauses (new variable y_1):

  $$(z_1 \vee z_2 \vee y_1) \wedge (z_1 \vee z_2 \vee \neg y_1)$$

- Every clause of the original formula that consists of 3 literals is taken as it is in the new formula.

- Finally, for every clause of the original formula that has more than 3 lterals, i.e $c = (z_1 \vee z_2 \vee \ldots z_k)$, construct the following clauses (y_i new variables):

  $$(z_1 \vee z_2 \vee y_1) \wedge (\neg y_1 \vee z_3 \vee y_2) \wedge (\neg y_2 \vee z_4 \vee y_3) \wedge \ldots \\ \wedge (\neg y_{k-4} \vee z_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee z_{k-1} \vee z_k)$$

If the length of the original Φ (size of the input) is $n \cdot m$ (m clauses with n at most literals each), then the length of formula Φ', will be $3m(n-2)$, i.e $m(n-2)$ clauses with 3 literals each; i.e., the length of Φ' will be $O(m \cdot n)$, polynomial with respect to the length of formula Φ and thus the transformation can be done in polynomial time. It remains to show that the original formula Φ has a satisfying truth assignment iff the new formula Φ' has a satisfying assignment (easy). □

Remark 2.3.2. Problem 2SAT belongs to P.

Another subproblem of SAT, that belongs to P, is *HORNSAT*. The given boolean formula consists of *Horn clauses*. We call *Horn clause* a clause that has at most one positive literal. i.e., all literals except possibly one , are negative, e.g:

$$(\neg x_2 \vee x_3), \quad (\neg x_1 \vee \neg x_2 \vee \neg x_3), \quad (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$$

The clauses having exactly one positive literal (as in the first and last example), are called *implications* because they can be written as : $(\neg x_2 \vee x_3) = (x_2 \rightarrow x_3)$, $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) = ((x_1 \wedge x_2 \wedge x_3) \rightarrow x_4)$. The algorithm solving **HORNSAT** is based exactly on this *implicational* form of the clauses. A Prolog program consists of Horn clauses.

2.4 Reduction of 3SAT to other problems

In general, when we reduce 3SAT to any other problem, our given data is a set of variables $u_1, \ldots, u_n$ and a set of clauses $c_1, \ldots, c_m$. Elements needed in our reduction :

- *Truthsetting*: Make sure that each variable has a unique truth value in all clauses.
- *Satisfaction*: Make sure that each clause contains at least one satisfiable literal.
- *Remaining (interconnections)-garbage collection*: Make sure we have a correct problem of the new type.

2.5 Reduction of 3SAT to VERTEX COVER

Theorem 2.5.1. *VERTEX COVER (VC) is NP-complete.*

Proof. It holds that $VC \in NP$. We will show that VC is NP-complete by reducing 3SAT to it ($3SAT \leq_m^p VC$).

We are given a set of clauses $C = \{c_1, \ldots, c_m\}$ with variables from a set of variables $U = \{u_1, \ldots, u_n\}$. We will construct a graph $G(V, E)$ and a positive $k \leq |V|$ such that G has a vertex cover of size $\leq k$ iff formula $(c_1 \wedge c_2 \wedge \ldots \wedge c_m)$ is satisfiable .

- For every variable $u_i \in U$ introduce 2 nodes in V, $u_i, \neg u_i$ and 1 edge in E, $(u_i, \neg u_i)$. That is, totally we get $2n$ nodes and n edges. This is truthsetting.
- For every clause $c_i \in C$ introduce 3 nodes in V, $c_1[i]$, $c_2[i]$, $c_3[i]$ and 3 edges in E, $(c_1[i], c_2[i])$, $(c_2[i], c_3[i])$, $(c_3[i], c_1[i])$. That is, totally we have $3m$ new nodes and $3m$ edges(this is satisfaction).
- Finally we add edges needed so that each triangle (satisfaction) is connected with 3 corresponding (truthsetting) literals and the new problem should be VERTEX COVER (remaining interconnections). For every node $c_k[i], 1 \leq k \leq 3$, we add the edge $(c_k[i], u_j)$ or $(c_k[i], \neg u_j)$ depending whether in the k-th position of clause c_i there is a literal u_j or $\neg u_j$ respectively. The number of new edges is $3m$ (3 for every clause).

The number of the nodes of the graph is $|V| = 2n + 3m$, and the number of edges is $|E| = n + 6m$. The size of the graph is polynomial with respect to the size of the given formula Φ ($3m$) and the construction can be done in polynomial time. We define $k = n + 2m$. We claim that the formula of the 3SAT problem is satisfiable iff there exists a vertex cover of size $\leq k$ in the graph $G(V, E)$. □

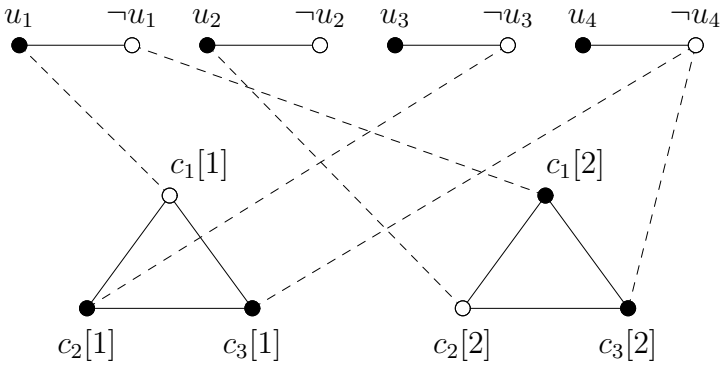

Figure 2.5: Reduction of 3SAT to VERTEX COVER

Example: We are given the formula

$$\Phi\colon (u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$$

Φ is satisfiable. e.g $t(u_1, u_2, u_3, u_4) = (T, T, T, T)$. Thus the graph $G(V, E)$ that is derived from this formula and is in figure 2.5, must have a vertex cover of size $\leq n + 2m = 4 + 2 \cdot 2 = 8$.

In the figure, the eight nodes that are black, form the vertex cover of the graph.

2.6 Reduction of 3SAT to 3D-MATCHING

Theorem 2.6.1. *3-DIMENSIONAL MATCHING is NP-complete.*

Proof. It is easy to prove 3DM $\in NP$, We will show 3SAT $\leq_m^p$ 3DM.

Given variables $U = \{u_1, \ldots, u_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$. We construct 3 disjoint sets W, X, Y with $|W| = |X| = |Y|$ and the set $M \subseteq W \times X \times Y$ such that M has a matching iff formula Φ: $c_1 \wedge c_2 \wedge \ldots \wedge c_m$ is satisfiable.

We put in W: $u_i[j], \neg u_i[j], 1 \leq j \leq m, 1 \leq i \leq n$ ($2mn$ of them).

We put in X:

- $a_i[j], 1 \leq j \leq m,\ 1 \leq i \leq n$, ($nm$ of them, used in truthsetting)
- $S_1[j], 1 \leq j \leq m$, (m of them , used in satisfaction)
- $g_1[k], 1 \leq k \leq (n-1)m$ ($(n-1)m$ of them , used in garbage collection).

Similarly to X, we construct set Y:

- $b_i[j], 1 \leq j \leq m,\ 1 \leq i \leq n$.
- $S_2[j], 1 \leq j \leq m$.
- $g_2[k], 1 \leq k \leq (n-1)m$.

We construt $M \subseteq W \times X \times Y$ like this: For each variable u_i we put in M the sets of triples T_i^t and T_i^f where:

$$T_i^t = \{(\neg u_i[j], a_i[j], b_i[j]), 1 \leq j \leq m\}$$
$$T_i^f = \{(u_i[j], a_i[j+1], b_i[j]), 1 \leq j \leq m\} \cup \{(u_i[m], a_i[1], b_i[m])\}$$

There are $2nm$ triples and they are used in truthsetting. For each clause c_j we put in M the set of triples C_j where:

$$C_j = \{(u_i[j], s_1[j], s_2[j]) \mid u_i \in c_j \text{ clause}\} \cup \{(\neg u_i[j], s_1[j], s_2[j]) \mid \neg u_i \in c_j \text{ clause}\}, 1 \leq j \leq m$$

In total there are $3m$ triples and they are used in satisfaction.

Finally, for garbage collection, we put in M the set of triples G where:

$$G = \{(u_i[j], g_1[k], g_2[k]),\ (\neg u_i[j], g_1[k], g_2[k])\}, \quad 1 \leq k \leq m(n-1),\ 1 \leq i \leq n, 1 \leq j \leq m$$

In total G has $2nm^2(n-1)$ elements.

Each matching $M' \subseteq M$ must contain at least nm triples from T_i^f, T_i^t so that all $a_i[j]$ and $b_i[j]$ are included. For each u_i, M' must contain either the whole T_i^f or the whole T_i^t. Matching M' gives the following satisfying assignment for Φ:

$$t(u_i) = \begin{cases} \text{True}, & T_i^t \subseteq M' \\ \text{False}, & T_i^f \subseteq M' \end{cases}$$

□

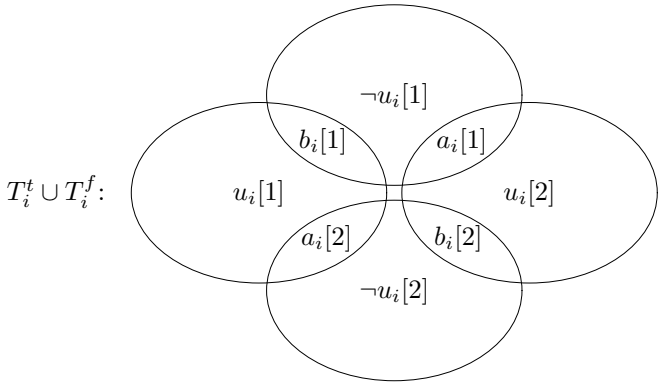

Figure 2.6: Reduction of 3SAT to 3DM: Truthsetting

Example 2.6.2. Given Φ: $(u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$ which is satisfiable by $t(u_1, u_2, u_3, u_4) = (T, T, F, T)$. For each variable u_i, M includes the set shown in figure 2.6. In addition, M includes sets C_1 and C_2 shown in figure 2.7.

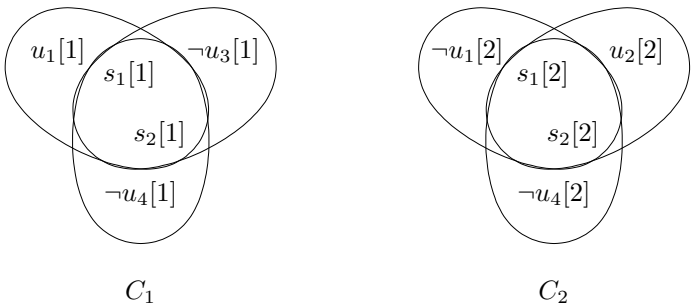

Figure 2.7: Reduction of 3SAT to 3DM: Satisfaction

A matching $M' \subseteq M$ that follows from assignment (T, T, F, T):

$$\begin{aligned} M' = \{&\neg u_1[1], a_1[1], b_1[1]), \\ &(\neg u_1[2], a_1[2], b_1[2]), \\ &(\neg u_2[1], a_2[1], b_2[1]), \\ &(\neg u_2[2], a_2[2], b_2[2]), \\ &(u_3[1], a_3[1], b_3[1]), \\ &(u_3[2], a_3[2], b_3[2]), \\ &(\neg u_4[1], a_4[1], b_4[1]), \\ &(\neg u_4[2], a_4[2], b_4[2])\} \bigcup T' \bigcup G' \end{aligned}$$

where

$$T' = \{(u_1[1], s_1[1], s_2[1]), (u_2[2], s_1[2], s_2[2])\}$$

and

$$\begin{aligned} G' = \{&(u_1[2], g_1[1], g_2[1]), (u_2[1], g_1[2], g_2[2]), \\ &(\neg u_3[1], g_1[3], g_2[3]), (\neg u_3[2], g_1[4], g_2[4]), \\ &(u_4[1], g_1[5], g_2[5]), (u_4[2], g_1[6], g_2[6])\} \end{aligned}$$

Remark 2.6.3. 2-DIMENSIONAL MATCHING is in P. The $2DM$ problem is also known as the (*marriage problem*). Other similar problems:

- 8 queens on a chessboard.
- 8 rooks on a chessboard.
- 8 rooks on a given allowed part of the chessboard.

2.7 Reduction of 3SAT to GRAPH 3-COLORABILITY

Theorem 2.7.1. *GRAPH 3-COLORABILITY is NP-complete.*

Proof. GRAPH 3-COLORABILITY is easily in NP.

We show $3SAT \leq_m^p GRAPH3-COLORABILITY$.

Given $U = \{u_1, \ldots, u_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$.

We construct a graph $G(V, E)$ such that formula Φ: $(c_1 \wedge c_2 \wedge \ldots \wedge c_m)$ is satisfiable iff G is 3-colorable. We construct G as follows:

- For each variable $u_i \in U$ we put in V the vertices u_i, $\neg u_i$, and in E the edges $(u_i, \neg u_i)$. We also put in V vertices a, b, and in E edges (a, b), (u_i, b), $(\neg u_i, b)$. The graph up to this point is shown in figure 2.8 (truthsetting).

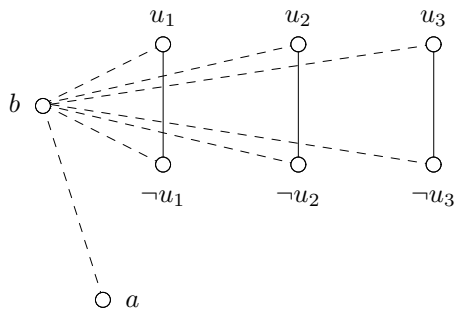

Figure 2.8: Reduction of 3SAT to 3-Colorability: Truthsetting

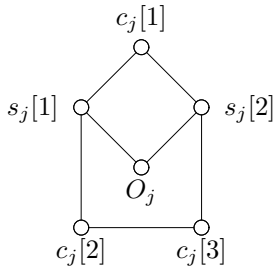

Figure 2.9: Reduction of 3SAT to 3 Colorability: Satisfaction

- For each clause c_j we augment graph G, with a subgraph shown in figure 2.9. We also put in E edges $(c_j[k], u_i)$ or $(c_j[k], \neg u_i)$ depending on whether the k-th literal of c_j is u_i or $\neg u_i$, respectively (satisfaction).

- Finally we add the remaining interconnections, i.e., we add edges (O_j, b) and (O_j, a).

In total we have $|V| = 2n + 6m + 2, |E| = 3n + 12m + 1$.

Φ is satisfiable iff $G(V, E)$ is 3-colorable.

If Φ is satisfiable, b gets color 2 and a gets color 1, so all O_j get color 0. We also color u_i, $\neg u_i$ as follows:

$$f(u_i) = \begin{cases} 1, & t(u_i) = True \\ 0, & t(u_i) = False \end{cases} \quad \text{and} \quad f(\neg u_i) = \begin{cases} 0, & t(u_i) = True \\ 1, & t(u_i) = False \end{cases}$$

All literals that are True in the satisfying assignment of Φ get color 1 and all literals that are False get color 0. Now, we need to color the remaining subgraphs, corresponding to clauses. Because Φ is satisfiable, there can be no subgraph colored like the one in figure 2.10. All possible cases are those shown in figure 2.11, (and symmetric to those).

Therefore, in all cases, G is 3-colorable.

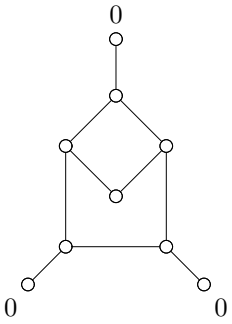

Figure 2.10: Subgraph of a non-satisfiable clause: not colorable

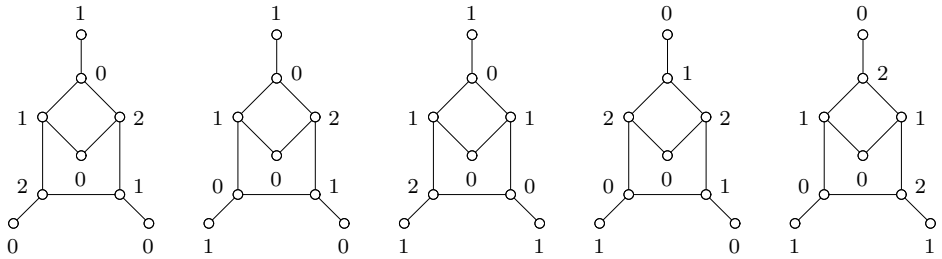

Figure 2.11: Possible colorings

Example: Given

$$\Phi\colon (u_1 \vee u_2 \vee \neg u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3)$$

and $t(u_1, u_2, u_3) = (T, T, T)$, $G(V, E)$ is shown in figure 2.12. The 3-coloring is shown in figure 2.13.

If G is 3-colorable then the formula is satisfiable. If not, there is a clause with all literals false, i.e., G has a subgraph like the one in figure 2.14, which is not 3-colorable; a contradictoin. □

2.8 Reduction of VERTEX COVER to HAMILTON CIRCUIT

Theorem 2.8.1. *HAMILTON CIRCUIT (HC) is NP-complete.*

Proof. HC can be easily shown to be in NP. We show that $VC \leq_m^p HC$.

Given a graph $G(V, E)$ and a positive integer $k \leq |V|$, we construct another graph $G'(V', E')$ such that $G(V, E)$ has a vertex cover of size $k \leq |V|$ iff $G'(V', E')$ has a hamilton circuit:

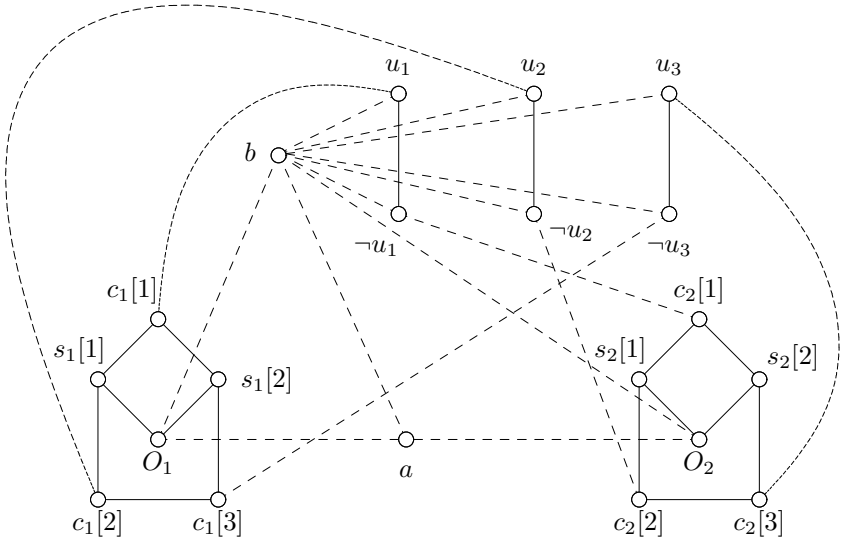

Figure 2.12: Corresponding graph

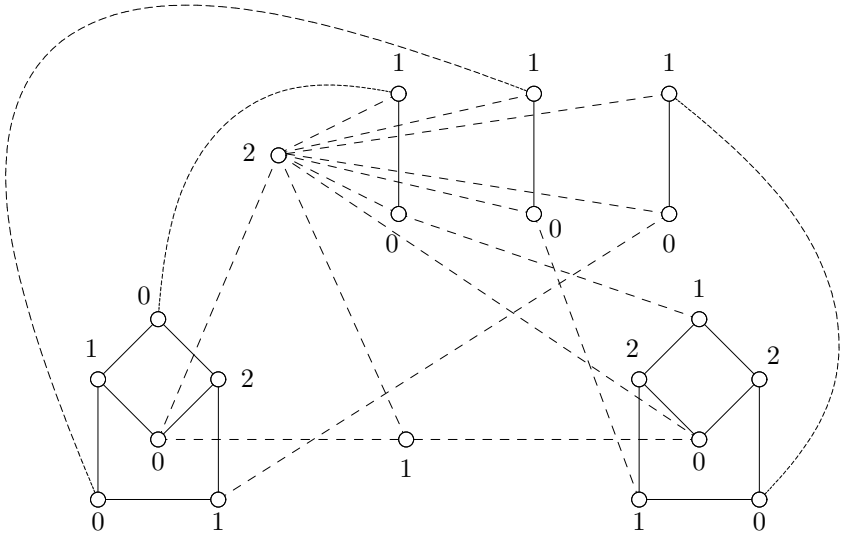

Figure 2.13: 3-coloring of the graph

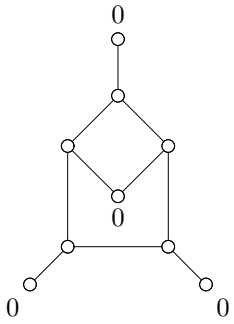

Figure 2.14: Subgraph that is not 3-colorable

- We put in V', k selector vertices a_1, a_2, ..., a_k.
- For each edge (u, v), we put in G' a "bridge", i.e., the component shown in figure 2.15.

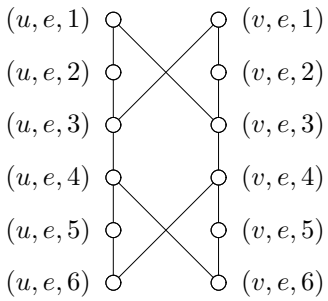

Figure 2.15: Bridge used in reduction of VC to HC

- $\forall v \in V$, we consider a permutation of edges that are incident to v:

$$\left\langle e_{v_{(1)}}, e_{v_{(2)}}, \ldots, e_{v_{(d(v))}} \right\rangle$$

where $d(v)$ is the degree of v. In total, we have n vertices and hence n such permutations.

For each vertex v, we connect all bridges that are in such a permutation, by adding the following edges to E':

$$E'_v = \{((v, e_{v_{(i)}}, 6), (v, e_{v_{(i-1)}}, 1))\}, 1 \leq i \leq d(v), \forall v \in V$$

Example: Given graph in figure 2.16, we choose the following permutations:

- For vertex 1: $< (1, 2), (1, 3), (1, 5) >$

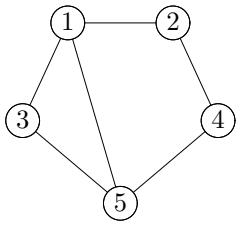

Figure 2.16: Example graph in reduction of VC to HC

- For vertex 2: $< (2,1), (2,4) >$
- For vertex 3: $< (3,1), (3,5) >$
- For vertex 4: $< (4,2), (4,5) >$
- For vertex 5: $< (5,1), (5,4), (5,3) >$

The n paths (and bridges) created are shown in figure 2.17.

Finally, we connect the extremities of each path with all the selector vertices a_1, a_2, ..., a_k, i.e., we add to E' the elements of set E'':

$$E'' = \{((v, e_{v_{(1)}}, 1), a_i), ((v, e_{v_{(d(v))}}, 6), a_i)\}, 1 \leq i \leq k, \forall v \in V$$

We show that $G(V, E)$ has a vertex cover of size $\leq k$ iff $G'(V', E')$ has a Hamilton cycle.

If G has vertex cover of size $\leq k$, then we denote the vertex cover with V_c. We will describe a hamilton circuit in G'.

Start at any a_i, w.l.o.g. at a_1 and then go to $(v, e_{v_{(1)}}, 1)$ where $v \in V_c$. To traverse the bridge starting at $(v, e_{v_{(1)}}, 1)$, by passing exactly one time at each of its points, choose one of the two ways shown in figure 2.18. If $u \notin V_c$, traverse the bridge like in (). If $u \in V_c$, traverse the bridge like in ().

Then go to the bridge starting at $(v, e_{v_{(2)}}, 1)$ corresponding to edge (v, u') of G. Use the same idea to traverse it (depending on whether $u' \in V_c$ or $u' \notin V_c$), etc. After traversing the last bridge of v go to another of the k selectors, say a_2.

From a_2 go to the testing components of another vertex in V_c, etc., until k such paths are traversed, finish at a_1 where the path started. This is a Hamilton cycle, because if there is a bridge that was not traversed, then the corresponding edge in G is not covered; a contradiction.

Also, if the k selectors are not enough to go through the testing components, then G has no vertex cover of size k; a contradiction.

If G' has a Hamilton cycle, then we have to go through k paths, so each bridge must be in some of these paths. So each edge of G (corresponding to

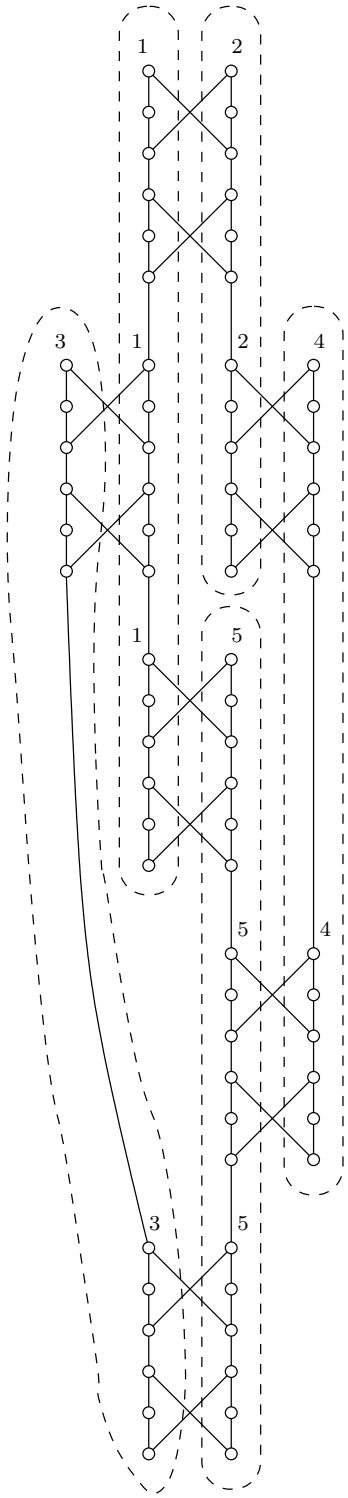

Figure 2.17: Bridges for the example graph

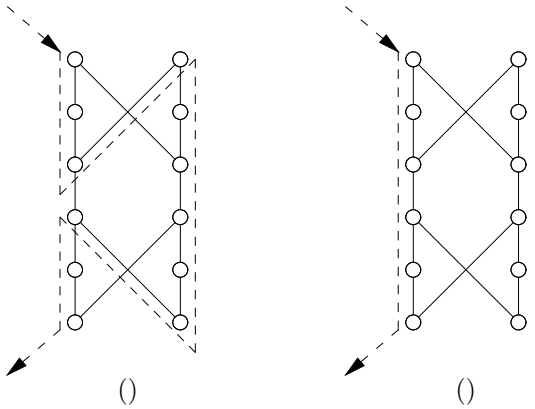

Figure 2.18: The two ways to traverse a bridge

a bridge in G') is covered by at least 1 vertex of G (corresponding to a path of bridges). □

2.9 Reduction of HAMILTON CIRCUIT to TRAVELING SALESPERSON PROBLEM

Theorem 2.9.1. *TRAVELING SALESPERSON PROBLEM (TSP) is NP-complete.*

Proof. TSP belongs to NP. We show that TSP is NP-complete by reducing HC to it ($HC \leq_m^p TSP$).

Given is a graph $G(V, E)$. We will construct a complete graph $G'(V', E')$ with weights $d(u, v), \forall (u, v) \in E'$ and a positive integer B, such that $G(V, E)$ will have a Hamilton cycle iff $G'(V', E')$ has a tour with total cost $\leq B$.

- To graph G we add all missing edges (complete graph). We assign weights to the edges G':

$$d(u, v) = \begin{cases} 1, & (u, v) \in G \\ 2, & (u, v) \notin G \end{cases}$$

- Finally, set $B = |V'| = |V|$.

The above construction can be done in polynomial time.

If G has a Hamilton cycle then this forms a tour in G' (goes once through every node and has total weight $B = |V'| = |V|$ because every edge has weight 1.

Conversely, if G' has a tour with total costs $\leq B = |V'| = |V|$, then this tour goes through $|V'|$ edges and consequently total costs are exactly $B = |V'|$. Thus every one of the $|V'|$ edges has weight 1. Thus all edges belong to G and G has a Hamilton cycle (the tour of G'). □

2.10 Reduction of VERTEX COVER to CLIQUE

Theorem 2.10.1. *CLIQUE is NP-complete.*

Proof. CLIQUE belongs to NP. We show that CLIQUE is NP-complete by reducing VERTEX COVER (VC) to it (VC $\leq_m^p$ Clique).

Given is a graph $G(V, E)$ and a positive integer $k \leq |V|$. We construct a graph $G'(V', E')$ and a positive integer $j \leq |V'|$ so that G has a vertex cover of size $\leq k$ iff G' has a clique $\geq j$.

- Let G' be the complementary graph of G. I.e., $V' = V$ and $E' = \{(u, v) \mid (u, v) \notin E\}$. Also, set $j = |V| - k = |V'| - k$.

This construction takes polynomial time.

If graph G has a vertex cover $V_c \subseteq V$ with $|V_c| \leq k$ then nodes $V - V_c$ are non adjacent, and thus adjacent in G'. G' has a clique of size $|V| - |V_c| \geq |V| - k = j$.

Conversely, if G' has a clique of size $\geq j = |V'| - k$ then in G there are at least $|V'| - k = |V| - k$ non-adjacent nodes. Thus the remaining (at most k) nodes are covering all edges of G (vertex cover of size $\leq k$). □

2.11 Reduction of CLIQUE to SUBGRAPH ISOMORPHISM

Theorem 2.11.1. *SUBGRAPH ISOMORPHISM is NP-complete.*

Proof. SUBGRAPH ISOMORPHISM belongs to NP. We show that SUBGRAPH ISOMORPHISM is NP-complete by reducing CLIQUE to it ($CLIQUE \leq_m^p SUBGRAPH\ ISOMORPHISM$).

Given is a graph G and a positive integer $k \leq |V|$. We construct two graphs $G_1(V_1, E_1)$ and $H(V_2, E_2)$ so that $G(V, E)$ has a clique of size $\geq k$ iff $G_1(V_1, E_1)$ has a subgraph isomorphic to $H(V_2, E_2)$. Set $G_1 = G$ and $H = K_k$.

If G has a clique with $V_c = k$, then G_1 has a subgraph isomorphic to H. Conversely, if G_1 has a subgraph isomorphic to H, then G has a clique of size at least k. □

2.12 Reduction of 3DM to PARTITION

Theorem 2.12.1. *PARTITION is NP-complete.*

Proof. PARTITION belongs to NP. We show that PARTITION is NP-complete by reducing 3-DIMENSIONAL MATCHING to it (3DM $\leq_m^p$ $PARTITION$).

Given are three disjoint sets W, X, Y with $|W| = |X| = |Y|$ and a set $M \subseteq W \times X \times Y$. We construct a set A and a weight function $w(a) \in Z^+, \forall a \in A$, so that set M has a matching $M' \subseteq M$ iff set A can be split into two equal weight subsets.

- Given are

$$\begin{aligned} W &= \{w_1, w_2, \ldots, w_q\} \\ X &= \{x_1, x_2, \ldots, x_q\} \\ Y &= \{y_1, y_2, \ldots, y_q\} \\ M &= \{m_1, m_2, \ldots, m_k\}, \quad \text{where } m_i = (w_r, x_l, y_n) \end{aligned}$$

  A will have $k+2$ elements. The first k elements a_1, a_2, ..., a_k correspond to m_i's. We will now define $w(a_i)$:

  – Every $w(a_i)$ is represented by a binary sequence that consists of $3q$ zones. Every zone has $p = \lceil \log_2(k+1) \rceil$ binary digits (figure 2.19). We label zones as follows:

$$w_1, w_2, \ldots, w_q \qquad x_1, x_2, \ldots, x_q \qquad y_1, y_2, \ldots, y_q$$

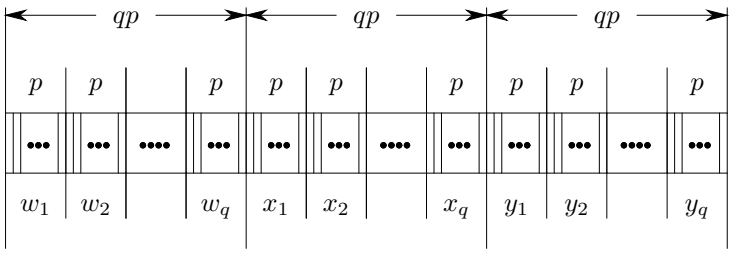

Figure 2.19: Binary sequence of the reduction of 3DM to PARTITION

If $m_i = (w_r, x_l, y_n)$, then in the corresponding zones w_r, x_l, y_n of $w(a_i)$ the rightmost bit is a 1 and all others are 0. Thus, the representation of $w(a_i)$ will contain exactly 3 1's, all other bits are 0. The decimal value of this binary number is:

$$w(a_i) = 2^{p(3q-r)} + 2^{p(2q-l)} + 2^{p(q-n)}.$$

The construction can be done in polynomial time.

– Here is the motivation for p equal to $\lceil \log_2(k+1) \rceil$: Even if all elements m_i have a common coordinate, adding all $w(a_i)$, will never create a carry for the next higher zone:

$$\begin{array}{cccccc} & & p & & & \\ 0 & 0 & \cdots & 0 & 1 & \\ & & \vdots & & & k \\ 0 & 0 & \cdots & 0 & 1 & \end{array}$$

– Set:

$$B = 2^{3qp-p} + 2^{3qpp-2p} + \ldots + 2^0 = \sum_{j=0}^{3q-1} 2^{pj}$$

I.e., the binary representation of B has a '1' at the rightmost position of each zone. Observe that in a matching, every coordinate is taken exactly once. Therefore:

$\exists A' \subseteq A\colon \sum_{a_i \in A'} w(a_i) = B \qquad \Longleftrightarrow$
$\exists M' \subseteq M\colon M'$ is a matching of M, where $M' = \{m_i \mid a_i \in A'\}$ $(*)$

- Finally, we define the weight of the last two elements of A. Element b_1 with weight $w(b_1) = 2\sum_{i=1}^{k} w(a_i) - B$ and element b_2 with weight $w(b_2) = \sum_{i=1}^{k} w(a_i) + B$ (polynomial construction also).

Now, we show that M has a matching iff A can be partitioned into two equal weight sets, A_1, A_2.

If M has a matching M', then because of $(*)$, $\exists A' \subseteq A\colon \sum_{a_i \in A'} w(a_i) = B$.

The total weight of A is:

$$W_{tot} = \sum_{i=1}^{k} w(a_i) + w(b_1) + w(b_2) = 4\sum_{i=1}^{k} w(a_i).$$

A_1 will consist of all $a_i \in A'$ and b_1. The weight is:

$$2\sum_{i=1}^{k} w(a_i) - B + B = 2\sum_{i=1}^{k} w(a_i).$$

Consequently, A_2 consisting of $a_i \in (A - A')$ and b_2 will have weight:

$$W_{tot} - 2\sum_{i=1}^{k} w(a_i) = 2\sum_{i=1}^{k} w(a_i).$$

Conversely, let A be partitioned into two equal weight sets A_1, A_2, each with weight $2\sum_{i=1}^{k} w(a_i)$. b_1, b_2 cannot both be in the same set A_j. A' contains those a_i that belong to the set A_i which contains b_1:

$$w(b_1) + \sum_{a_i \in A'} w(a_i) = 2\sum_{i=1}^{k} w(a_i) \Rightarrow \sum_{a_i \in A'} w(a_i) = B$$

Because of $(*)$, this means that: $\exists M' \subseteq M\colon M' = \{m_i \mid a_i \in A'\}$ and M' is a matching. □

2.13 Reduction of PARTITION to DISCRETE KNAPSACK

Theorem 2.13.1. *DISCRETE KNAPSACK is NP-complete.*

Proof. DKNAPSACK belongs to NP. We show that DKNAPSACK is NP-complete by reducing PARTITION to it ($PARTITION \leq_m^p DKNAPSACK$).

Given a set A and a weight function $w(a_i), \forall a_i \in A$, we construct a set U, a weight function $w'(u_i), \forall u_i \in U$, and a cost function $p(u_i), \forall u_i \in U$ and two positive integers W, P so that A can be partitioned into two equal weight sets iff there exists: $U' \subseteq U\colon \sum_{u \in U'} w'(u) \leq W$, $\sum_{u \in U'} p(u) \geq P$.

- Set $U = A$
- The weights $w'(u), \forall u \in U$ are the same as $w(a), \forall a \in A$
- The costs $p(u), \forall u \in U$ are also the same as $w(a), \forall a \in A$
- Set
$$W = P = \frac{1}{2}\sum_{a \in A} w(a) = \frac{1}{2}\sum_{u \in U} w'(u)$$

The construction can be done in polynomial time.

If A can be partitioned into two equal weight subsets, A' and $A - A'$, because the total weight of A is $\sum_{a \in A} w(a)$, the weight of a A' is $\frac{1}{2}\sum_{a \in A} w(a)$. Thus for $U' = A'$: $\sum_{u \in U'} w'(u) = W \leq W$ and $\sum_{u \in U'} p(u) = P \geq P$.

Conversely, $\exists U' \subseteq U\colon \sum_{u \in U'} w'(u) \leq W$ and $\sum_{u \in U'} p(u) \geq P$. But $\sum_{u \in U'} w'(u) = \sum_{u \in U'} p(u) = B = P = \frac{1}{2}\sum_{u \in U} w'(u)$. Consequently, for $A' = U'$, A is partitioned into two equal weight subsets, A' and $A - A'$. □

Chapter 3

Complexity classes

3.1 Basic definitions

The computational model: Turing machine (deterministic or non deterministic).

Definition 3.1.1. TIME($t(n)$) (or DTIME($t(n)$)): problems that can be solved by a *deterministic* TM in time $t(n)$.

Definition 3.1.2. NTIME($t(n)$): problems that can be solved by a *non deterministic* TM in time $t(n)$.

Definition 3.1.3. SPACE($s(n)$) (or DSPACE($s(n)$)): problems that can be solved by a *deterministic* TM by using additional working space $s(n)$.

Definition 3.1.4. NSPACE($s(n)$): problems that can be solved by a *non deterministic* TM by using additional working space $s(n)$.

Based on the above, we define:

- P = PTIME = $\bigcup_{i\geq 1}$ DTIME(n^i)
- NP = NPTIME = $\bigcup_{i\geq 1}$ NTIME(n^i)
- PSPACE = $\bigcup_{i\geq 1}$ DSPACE(n^i)
- NPSPACE = $\bigcup_{i\geq 1}$ NSPACE(n^i)
- L = DSPACE($\log n$)
- NL = NSPACE($\log n$)
- DEXP = $\bigcup_{i\geq 1}$ DTIME(2^{n^i})

- DEXPSPACE $= \bigcup_{i \geq 1} \text{DSPACE}(2^{n^i})$
- f is *constructible*: there is a TM which, $\forall$ input x with $|x| = n$, accepts the input in time $O(n + f(n))$ (time-constructible) or working space $O(f(n))$ (space-constructible).

Proposition 3.1.5. *If f is constructible, then:*

- $\text{DSPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
- $\text{DTIME}(f(n)) \subseteq \text{NTIME}(f(n))$
- $\text{DTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$
- $\text{NTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$
- *If $f(n) > \log n$ then*
  - $\text{NTIME}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$
  - $\text{DSPACE}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$
- $\text{NSPACE}(f(n)) \subseteq \text{DTIME}(k^{\log n + f(n)})$

Theorem 3.1.6 (Savitch's Theorem, 1970)**.** *If $f(n) \geq \log n$ then*

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$$

An immediate consequence: PSPACE = NPSPACE.

Thus:

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE}$$

We know that: L $\neq$ PSPACE and NL $\neq$ PSPACE.

Open problems:

$$\text{L} \supseteq \text{NL} \supseteq \text{P} \supseteq \text{NP} \supseteq \text{PSPACE}$$

See also figure 3.1.

Functional complexity classes:

Definition 3.1.7. FP = functions that can be computed by a deterministivc TM in polynomial time

Definition 3.1.8. FL = functions that can be computed by a deterministivc TM in logarithmic space

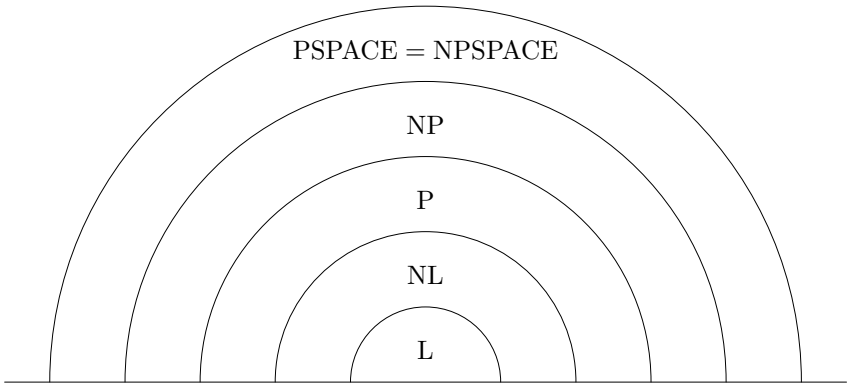

Figure 3.1: Complexity classes

3.2 Hierarchy theorems

For a deterministic TM with at least three tapes:

Theorem 3.2.1 (Fürer, 1982)**.** *Given $t_2(n) > n$, there is a language accepted in time t_2, but not in time t_1 for any $t_1 = o(t_2(n))$.*

Theorem 3.2.2 (Hartmanis, Lewis, Stearns, 1965)**.** *Given $s_2(n) > \log n$, there is a language accepted in space s_2, but not in space s_1 for any $s_1 = o(s_2(n))$.*

We insist on constructible complexity functions, because otherwise we have:

Theorem 3.2.3 (Gap theorem)**.** *There is a recursive function $t(n)$, such that* $\mathrm{TIME}(t(n)) = \mathrm{TIME}(2^{t(n)})$.

3.3 Complementary complexity classes

Complement of a language: $\overline{L} = \{x \mid x \notin L\}$.
Complement class:

$$\mathrm{co}\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}.$$

For example, $\overline{\mathrm{SAT}} \in \mathrm{coNP}$, and P is closed under complement.

Theorem 3.3.1 (Immerman-Szelepcsényi)**.** $\mathrm{NSPACE}(s(n))$ *is closed under complement.*

3.4 Reductions

Definition 3.4.1 (Karp reduction)**.**

$$A \leq_m^P B\colon \quad \exists f \in \mathrm{FP}, \forall x(x \in A \iff f(x) \in B)$$

Definition 3.4.2 (Log-space reduction)**.**

$$A \leq_m^L B\colon \quad \exists f \in \mathrm{FL}, \forall x(x \in A \iff f(x) \in B)$$

We have $A \leq_m^L B \implies A \leq_m^P B$, but not the converse.

Definition 3.4.3. Class C is *closed* under reduction $\leq$ if

$$A \leq B \wedge B \in C \implies A \in C.$$

Definition 3.4.4 (Hardness)**.** A is C-hard, under $\leq$, if:

$$\forall B \in C : B \leq A.$$

Definition 3.4.5 (Completeness)**.** A is C-complete, under $\leq$, if:

$$A \text{ is } C - \text{hard under} \leq \quad \wedge \quad A \in C.$$

Complete problems for several complexity classes:

For NL, we have Reachability (under log-space reductions). For P, we have: Circuit-Value and Linear Programming (again under log-space reductions). For NP, we have 3SAT. For PSPACE, we have QBF (Quantified Boolean Formula satisfiability problem). For EXP, we have $n \times n$ Go. For EXPSPACE, we have RegExp$(\cup, \cdot, *, ^2)$, i.e., checking equivalence of regular expressions, containing the following operators: $\cup$ (union), $\cdot$ (concatenation), $*$ (Kleene star) and ^{2}, where $\alpha^2 = \alpha \cdot \alpha$.

3.5 Parameters for defining complexity classes

- model of computation: Turing machine (TM), Random Access Machine (RAM), finite automaton, Linearly Bounded Automaton (LBA), parallel RAM (PRAM), monotone circuits;
- operation mode: deterministic, non deterministic, probabilistic, alternating, parallel;
- computation kind: decider, acceptor, generator, transducer;

- resources: number of computation steps, number of comparisons, number of multiplications, time, memory space, number of processors, number of alternations in computation tree, size of circuit, depth of circuit;
- other tools: randomness, oracles, interactivity, promise, operators;
- bounds with respect to the size of input: e.g., $O(n^3)$ or polynomial, time/space $(t(n), s(n))$ tradeoff, Probabilistic Checkable Proofs: PCP$(r(n), q(n))$ (using $r(n)$ random bits and $q(n)$ queries to the proof).

3.6 Computational tree model for TMs

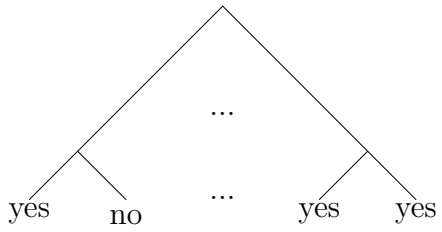

Figure 3.2: Computational tree model

We use quantifiers $(\exists, \forall)$ over computation paths (from the root to a leaf) in computation trees. E.g., we write $\exists y$, instead $\exists y\colon |y| \leq p(|x|)$, where y: variable for paths, x: variable for inputs, p: polynomial.

In all cases, below, we use a predicate R computed in polynomial time.

$$L \in \mathrm{P} \iff \exists R \in \mathrm{P}\colon \begin{cases} x \in L \implies \forall y R(x,y) \\ x \notin L \implies \forall y \neg R(x,y) \end{cases}$$

$$L \in \mathrm{NP} \iff \exists R \in \mathrm{P}\colon \begin{cases} x \in L \implies \exists y R(x,y) \\ x \notin L \implies \forall y \neg R(x,y) \end{cases}$$

$$L \in \mathrm{coNP} \iff \exists R \in \mathrm{P}\colon \begin{cases} x \in L \implies \forall y R(x,y) \\ x \notin L \implies \exists y \neg R(x,y) \end{cases}$$

The two quantifiers used in "$x \in L$" and "$x \notin L$" define fully the complexity class, so we use the following notation:

$$\mathrm{P} = (\forall, \forall), \quad \mathrm{NP} = (\exists, \forall), \quad \mathrm{coNP} = (\forall, \exists).$$

Chapter 4

Randomness, Oracles, Parallelism

4.1 Randomness

In the computation tree model, we assume that non-deterministic choices at each node are done with probability 1/2. To express that an *'overwhelming' majority* of computation paths has some property, we use a new quantifier: $\exists^+$.

Thus, we define BPP (Bounded error Probabilistic Polynomial):

Definition 4.1.1 (BPP $= (\exists^+, \exists^+)$)**.**

$$L \in \mathrm{BPP} \iff \exists R \in \mathrm{P}\colon \begin{cases} x \in L \implies \exists^+ y R(x,y) \\ x \notin L \implies \exists^+ y \neg R(x,y) \end{cases}$$

The exact definition of overwhelming majority is not important as long as it is bounded over 1/2. It can be bigger than any of the following: $1/2+\varepsilon$, $1/2 + 1/p(|x|)$, 2/3, 99%, $1 - 2^{-p(|x|)}$ (where $p(|x|) > 1$).

BPP algorithms are also known as Monte Carlo or two-sided error. BPP is closed under complement.

For one-sided error, we have:

Definition 4.1.2 (RP $= (\exists^+, \forall)$)**.**

$$L \in \mathrm{RP} \iff \exists R \in \mathrm{P}\colon \begin{cases} x \in L \implies \exists^+ y R(x,y) \\ x \notin L \implies \forall y \neg R(x,y) \end{cases}$$

We can define the complement of RP in a similar way: coRP = $(\forall, \exists^+)$:

$$L \in \text{coRP} \iff \exists R \in \text{P}: \begin{cases} x \in L \implies \forall y R(x,y) \\ x \notin L \implies \exists^+ y \neg R(x,y) \end{cases}$$

We have RP $\subseteq$ BPP, coRP $\subseteq$ BPP, but RP = coRP is open.

ZPP = RP $\cap$ coRP (Zero error Probabilistic Polynomial) Alternatively, a ZPP algorithm can give three answers in each run: 'yes', 'no', and 'I do not know'.

ZPP algorithms are also known as Las Vegas.

We do not know if there exist complete problems for the classes BPP, RP, ZPP).

If we don't bound the error away from 1/2, we use the quantifier $\exists_{1/2}$.

For unbounded two-sided error, we have class PP (Probabilistic Polynomial):

Definition 4.1.3 (PP = $(\exists_{1/2}, \exists_{1/2})$)**.**

$$L \in \text{PP} \iff \exists R \in \text{P}: \begin{cases} x \in L \implies \exists_{1/2} y R(x,y) \\ x \notin L \implies \exists_{1/2} y \neg R(x,y) \end{cases}$$

Proposition 4.1.4. NP $\subseteq$ PP.

RL (Randomized Logspace) contains problems that have one-sided error algorithm that uses logarithmic space and polynomial number of bits w.r.t input size.

4.2 Polynomial Hierarchy

Computation with an oracle: An algorithm uses an oracle for problem Π, if the algorithm can pose (during the computation) a query to the oracle for some instance x of problem Π, whether $x \in \Pi$, and the oracle answers 'yes' or 'no'. The algorithm does not spend any additional resources when it asks a question to the oracle (it gets the correct answer for 'free' even though problem Π can be very hard).

Definition 4.2.1 (Oracle classes)**.**

- $\mathcal{C}^{\Pi}$: class of problems solvable by an algorithm in class $\mathcal{C}$ that uses an oracle for problem Π
- $\mathcal{C}^{\mathcal{C}_o} = \bigcup_{\Pi \in \mathcal{C}_o} \mathcal{C}^{\Pi}$

E.g.: $\mathrm{P}^{\mathrm{SAT}}$: Problems solvable by a deterministic polynomial algorithm using an oracle for the SAT problem. Another equivalent description: P^{NP} (because SAT is NP-complete).

Classes in the polynomial hierarchy:

Definition 4.2.2. ($k \geq 0$)

- $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathrm{P}$
- $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$, $\Pi_{k+1}^p = \mathrm{co}\Sigma_{k+1}^p$, $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p}$, $\Delta\Sigma_k^p = \Sigma_k^p \cap \Pi_k^p$
- Polynomial hierarchy: $\mathrm{PH} = \bigcup_{k \in \mathbb{N}} \Sigma_k^p$

$\Sigma_1^p = \mathrm{NP}$, $\Pi_1^p = \mathrm{coNP}$ and $\forall k \geq 0$: $\Sigma_k^p \subseteq \Sigma_{k+1}^p$ and $\Pi_k^p \subseteq \Sigma_{k+1}^p$. Although there is no proof of the strictness of the above inclusions (like in the arithmetic hierarchy), it is believed that the polynomial hierarchy is strict. If PH is not strict, then $\exists k$: $\mathrm{PH} = \Sigma_k^p$, i.e., *PH collapses at the k-th level.*

A different way to define PH: using quantifier alternation ($\exists$ and $\forall$). In all cases, quantifiers quantify over strings whose size is bounded by a polynomial p w.r.t. input size.

Proposition 4.2.3. *$L \in \Sigma_k^p$ iff $\exists$ predicate R computable in polynomial time and a polynomial p that bounds the quantified variables, such that:*

$$x \in L \iff \exists y_1 \forall y_2 \ldots Q y_k \; R(x, y_1, y_2, \ldots, y_k),$$

where $Q = \begin{cases} \exists, & k\text{: odd} \\ \forall, & k\text{: even} \end{cases}$.

Similarily for Π_k^p, but the quantifiers start from $\forall$:

Proposition 4.2.4. *$L \in \Pi_k^p$ iff $\exists$ predicate R computable in polynomial time and a polynomial p that bounds the quantified variables, such that:*

$$x \in L \iff \forall y_1 \exists y_2 \ldots Q y_k \; R(x, y_1, y_2, \ldots, y_k),$$

where $Q = \begin{cases} \forall, & k\text{: odd} \\ \exists, & k\text{: even} \end{cases}$.

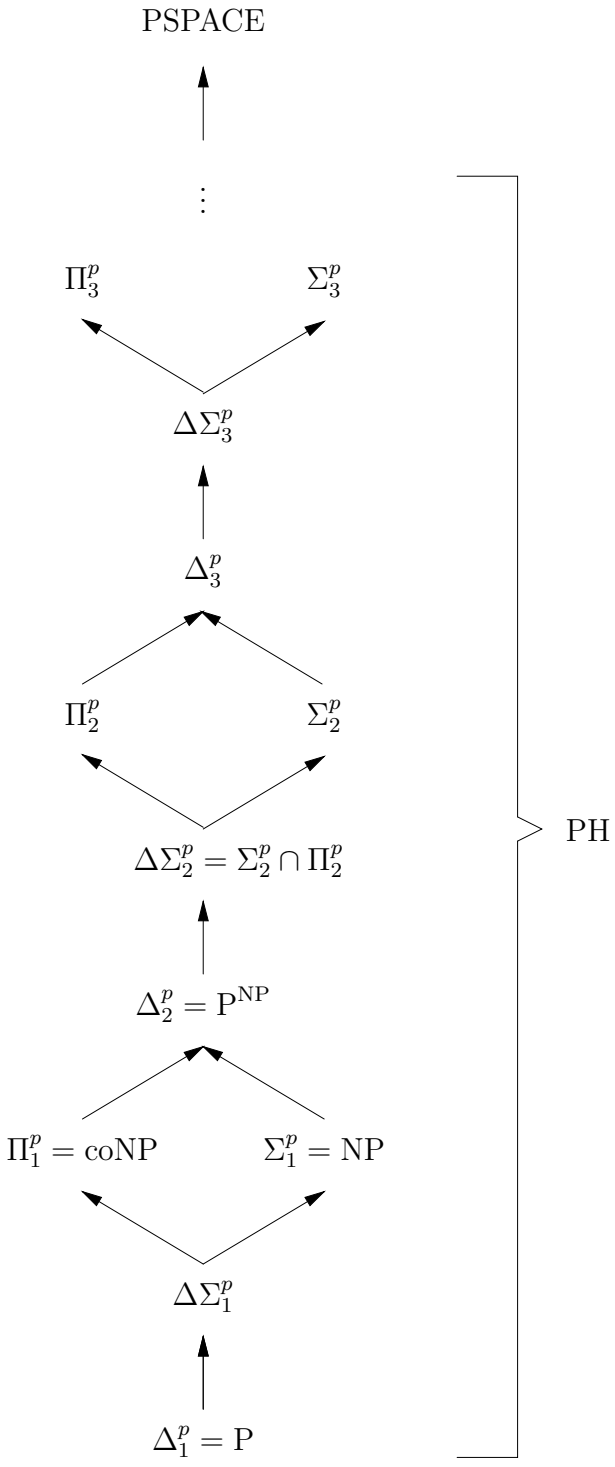

Figure 4.1: Polynomial hierarchy PH

4.2.1 Alternating TMs

Consider the tree representation of the computation of an NPTM: the answer is 'yes' if there is at least one leaf saying 'yes'. Every node in the tree has answer value namely the disjunction ($\vee$) of the answers of its children.

A coNPTM accepts if all leaves say 'yes', so every node has an answer value of the the conjuction ($\wedge$) of its children.

All nodes in an NPTM tree are of type $\vee$, or $\exists$, or existenstial. All nodes in a coNPTM tree are of type $\wedge$, or $\forall$, or universal.

In an *alternating TM* the corresponding computation tree can have internal nodes of types $\vee$ and $\wedge$. The number of *type alternations* is important. The maximum *number* of *alternations* allowed along a path of the computation tree (this number is usually bounded) is an indication of the capabilities of the respective TM.

For example, the tree of figure 4.2 has two alternations.

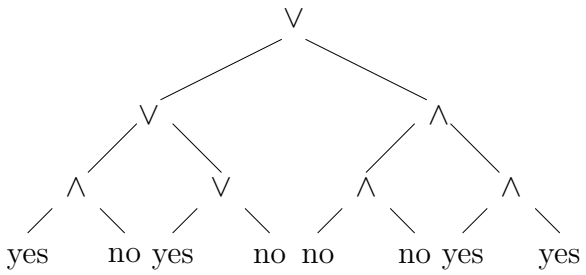

Figure 4.2: Computation tree with alternations

It can be shown that the polynomial hierarchy is exactly the class of languages accepted by TMs with bounded number of alternations.

- $L \in \Sigma_k^p$ iff L is acceptable by a TM with at most k alternations, starting with type $\vee$ (at the root);
- $L \in \Pi_k^p$ iff L is acceptable by a TM with at most k alternations, starting with type $\wedge$ (at the root).

4.3 Parallelizable problems – Circuits

A circuit is a directed acyclic graph[1], with a set of *input nodes* and one *output node*. Inputs to the circuit are truth values (0 or 1) and each internal node corresponds to some boolean function (gate) with possibly many inputs.

[1] Acyclicity means no circuits with feedback are considered.

If a circuit C has n 1-bit inputs: $x_1, x_2, \ldots, x_n$, then $\forall x \in \{0,1\}^n$, C computes a unique value at the output, namely $C(x)$. If $C(x) = 1$, C accepts the n-bit input x.

A circuit (because of its fixed nature) can only answer for inputs of size exactly n, but a TM (or an algorithm in general) can process inputs of any size. For this reason, we consider a family of circuits $\{C_1, C_2, \ldots\}$, where each C_n has n input nodes.

Language accepted by a circuit family:

$$L(C) = \{x \mid C_{|x|}(x) = 1\}.$$

Remark: Circuit families, in contrast to TMs, are not countable! To overcome this problem, we consider only uniform circuit families. For a uniform circuit family there is an efficient algorithm that constructs a representation of C_n, for every n. E.g.: for P-uniform families there exist polynomial construction algorithms. However for classes of lower complexity than P a more restricted kind of uniformity is used:

Definition 4.3.1. A family of circuits is DLOGTIME-uniform if $\exists$ TM (with random access capability to its input tape) that answers the following questions in $O(\log n)$ time:

- is there an edge from node u to node v in C_n?
- What kind of gate node u is?

Size of a circuit is the number of nodes in the corresponding graph (an indication of the cost of such a circuit: number of gates). However, size is not a good indication of the time it takes for a circuit to compute, because many gates function in parallel. When time is considered, the *depth* of the circuit is more important (depth defined as the length of the longest path from an input to the output node).

The kind of gates allowed is also important.

1. Gates with bounded in-degree (bounded fan-in) (including unary $\neg$ gates). It suffices to consider only binary $\wedge$ and $\vee$ gates (along with unary $\neg$ gates).
2. $\wedge$ and $\vee$ gates with unbounded in-degree (unbounded fan-in) (including unary $\neg$ gates).
3. Threshold gates[2] with unbounded in-degree, and unary $\neg$ gates. It suffices to use majority gates, which output 1 iff at least $r/2$ of the r inputs are 1, and unary $\neg$ gates.

[2]Threshold gates are used in neural networks.

Circuit classes:

Definition 4.3.2. ($k \geq 0$):

1. NC^k: class of languages acceptable by DLOGTIME-uniform circuit families of polynomial size and $O(\log^k n)$ depth, with gates of the first kind (bounded fan-in).

2. AC^k: class of languages acceptable by DLOGTIME-uniform circuit families of polynomial size and $O(\log^k n)$ depth, with gates of the second kind (unbounded fan-in).

3. TC^k: class of languages acceptable by DLOGTIME-uniform circuit families of polynomial size and $O(\log^k n)$ depth, with gates of the third kind (threshold gates).

4. SC^k: class of languages acceptable by a DTM in polynomial time and in $O(\log^k n)$ space.

$\mathrm{NC} = \bigcup_{k \in \mathbb{N}} \mathrm{NC}^k$. This is Nick's Class, in honor of Nicholas Pippenger, who was one of the first to investigate circuit complexity.

Many other models of parallel computation (e.g., CRAM), apart from circuits, can be used to define NC, which is an indication of the robustness of the class and its close relation to parallelizable problems.

The 'A' in AC^k stands for 'alternation', because AC^k, for $k \geq 1$, contains exactly the languages acceptable by a TM in $O(\log n)$ space with at most $O(\log^k n)$ alternations. The 'T' in TC^k stands for 'threshold'. SC or "Steve's Class", in honor of Steve Cook.

Relations between parallel complexity classes:

Theorem 4.3.3. $\forall\ k \geq 0$, $\mathrm{NC}^k \subseteq \mathrm{AC}^k \subseteq \mathrm{TC}^k \subseteq \mathrm{NC}^{k+1}$.

Relations to other classes:

Theorem 4.3.4. Regular $\subseteq \mathrm{NC}^1 \subseteq \mathrm{L} = \mathrm{SC}^1 \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$.
Regular $\subset$ CF $\subset \mathrm{AC}^1$.

I.e., checking if a string is produced by a given context free language is in NC^2.

Chapter 5

Interactivity, PCP

5.1 Interactivity

In this section we define complexity classes with the help of two TMs that interact by exchanging messages. Usually, one of them, the Prover, tries to convince the other one, the Verifier, that a string belongs to some language. A tool that is used is *randomness*.

5.1.1 Interactive proof systems (IP)

Consider a prover, trying to convince a verifier about the truth of a statement of the form '$x \in L$'. The prover is omnipotent, i.e., an algorithm that has no bounds on the resources it can use (time, space). On the contrary, the verifier is just a polynomial time probabilistic algorithm. The prover and the verifier take part in a communication protocol (they send messages to each other). Depending on the messages V receives from P, V accepts P's proof, otherwise V rejects. The prover may be dishonest, and attempt to convince V that '$x \in L$', even for x such that '$x \notin L$'. The verifier (against the omnipotent prover) can use polynomial time, but principally randomness. The class IP was defined by Goldwasser, Micali, Rackoff.

Definition 5.1.1. $L \in$ IP:

- $x \in L \implies \exists$ prover P, such that verifier V always accepts (i.e., with probability 1)
- $x \notin L \implies \forall$ prover P, verifier V does not accept, with overwhelming probability.

Consider the *graph non isomorphism* problem: "Given two graphs, are they non isomorphic?". This problem is in coNP.[1] We give a protocol for the graph non isomorphism problem, showing it is in IP.

Initially, the verifier has the two graphs G_1 and G_2. V chooses randomly one of the two, say G_i, and computes a random isomorphic graph to G_i, say H (this can be done by choosing a random permutation of the n vertices of graph G_i). V sends H to the prover, and asks for a j such that G_j is isomorphic to H. P responds with a $j \in \{1, 2\}$. V accepts if $i = j$, otherwise V rejects.

If G_1, G_2 are really non isomorphic, then P, being omnipotent, finds the (unique) graph G_i which is isomorphic to the H that V sent, and gives the correct answer to V, thus V accepts. If however G_1, G_2 are isomorphic, P cannot infer from which G_i H stems from, so P can do nothing better than choose randomly from $\{1, 2\}$. Thus, if the graphs are isomorphic V rejects with probability $1/2$.

Any $L \in$ PH has an IP protocol. In fact, an even stronger result holds:

Theorem 5.1.2 (Shamir)**.** IP = PSPACE

What if the verifier can interact with more than one provers? If the provers can communicate with each other, then we still get class IP (a prover, being an omnipotent algorithm, can simulate many other provers). However, if provers cannot communicate between themselves, then we have the more powerful class MIP (Multi IP). In fact: MIP = NEXP.

5.1.2 Arthur-Merlin classes

In class IP the verifier 'hides' the random bits used. In fact, this is a necessary ingredient of the proof that graph non isomorphism is in IP. It seems that if the verifier is forced to reveal the random bits used, we would get a less powerful class than IP. In this class, the prover is called Merlin and the verifier is called Arthur (description due to Babai). In fact, we can even require that Arthur's messages are less restricted: he just sends his random bits to Merlin. Depending on Merlin's answers, Arthur decides whether he accepts or rejects.

We say that Arthur and Merlin play a k move game (every move corresponds to a message): if Arthur moves first the game is denoted by AM(k), and if Merlin moves first it is denoted by MA(k). E.g., AM(1) = A, AM(2) = AM, AM(3) = AMA, MA(1) = M, MA(2) = MA, MA(3) = MAM.

[1] The complementary problem, graph isomorphism, is in NP, but it does not seem to be NP-complete.

Definition 5.1.3. $L \in \mathrm{AM}(k)$ iff $\exists$ a k move game where Arthur plays first and:

- $x \in L \implies$ Arthur is convinced with probability greater than $2/3$ that $x \in L$
- $x \notin L \implies$ Arthur is convinced with probability less than $1/3$ that $x \in L$.

Using generalized quantifiers, these classes can be defined as (Zachos):

$$\mathrm{AM} = \mathrm{AM}(2) = (\exists^+\exists, \exists^+\forall), \quad \mathrm{MA} = \mathrm{MA}(2) = (\exists\exists^+, \forall\exists^+),$$

and for even k, if $\mathrm{AM}(k) = (\mathbf{Q}_1, \mathbf{Q}_2)$, where $\mathbf{Q}_1$, $\mathbf{Q}_2$ sequences of quantifiers:

$$\mathrm{AM}(k+1) = (\mathbf{Q}_1\exists^+, \mathbf{Q}_2\exists^+), \quad \mathrm{AM}(k+2) = (\mathbf{Q}_1\exists^+\exists, \mathbf{Q}_2\exists^+\forall).$$

This description can be simplified (Zachos):

$$\mathrm{AM} = \mathrm{AM}(2) = (\forall\exists, \exists^+\forall), \quad \mathrm{MA} = \mathrm{MA}(2) = (\exists\forall, \forall\exists^+),$$

and for even k, if $\mathrm{AM}(k) = (\mathbf{Q}_1, \mathbf{Q}_2)$, where $\mathbf{Q}_1$, $\mathbf{Q}_2$ sequences of quantifiers:

$$\mathrm{AM}(k+1) = (\mathbf{Q}_1\forall, \mathbf{Q}_2\exists^+), \quad \mathrm{AM}(k+2) = (\mathbf{Q}_1\forall\exists, \mathbf{Q}_2\exists^+\forall).$$

Using properties of quantifiers:

Proposition 5.1.4. $\mathrm{MA} \subseteq \mathrm{AM}$.

Proposition 5.1.5. *The Arthur-Merlin hierarchy collapses, i.e.:*

$$\mathrm{AM} = \mathrm{AM}(k) = \mathrm{MA}(k+1), \quad \forall\ k \geq 2.$$

Although, the Arthur-Merlin class with polynomial number of messages seems weaker (because of the public random bits) than IP, in fact, Goldwasser and Sipser proved that they are equivalent.

5.2 Probabilistically Checkable Proofs — PCP

If we replace (in interactive proof systems) the prover by a simple proof, we get class PCP. Equivalently, in PCP, the prover sends only one communication message; he sends the whole proof to V. Such a proof is checked probabilistically by V.

Definition 5.2.1. $L \in$ PCP:

- $x \in L \implies \exists$ proof Π such that the verifier V always accepts (i.e., with probability 1)
- $x \notin L \implies \forall$ 'proofs' Π, the verifier V does not accept, with overwhelming probability.

This class seems a lot more powerful than IP because in PCP the verifier does not have to 'confront' an adaptable prover, but a static proof. And in fact: PCP = MIP(= NEXP). That is why we consider restrictions of PCP. We consider two kinds of resources for which the verifier has some restrictions in their use:

- randomness (in the form of random bits);
- bits of the proof that the verifier checks (queries to the proof).

Definition 5.2.2. PCP$(r(n), q(n))$ consists of languages $L \in$ PCP such that the polynomial time verifier V uses $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof.

Relation to other classes: PCP = PCP(poly(n), poly(n)), P = PCP$(0, 0)$, NP = PCP$(0,$ poly$(n))$, coRP = PCP(poly$(n), 0)$.

A very important result (Arora, Lund, Motwani, Sudan, Szegedy):

Theorem 5.2.3 (PCP)**.** NP = PCP$(\log n, 1)$.

An application of the PCP theorem is in proofs of non-approximability (see section 6.2).

The basic tool in the proof of the PCP theorem is a method (PCP encoding) that smears a possible mistake in a proof in all parts of the proof, so that the verifier has an overwhelming probability to find the error. This method is based on techniques of error correcting codes.

Chapter 6

Counting, Approximation

6.1 Counting classes

Counting classes are defined based on the number of solutions of a problem. They are function classes (like FP).

Definition 6.1.1. #P: the class of functions f for which there is a polynomial time NDTM, whose computation tree has exactly $f(x)$ accepting computation paths (for input x).

Definition 6.1.2. #L: the class of functions f for which there is a logarithmic space NDTM, whose computation tree has exactly $f(x)$ accepting computation paths (for input x).

Reductions that preserve the number of solutions are useful for counting classes.

A complete problem for #P is #SAT: "Given a CNF formula, how many different satisfying truth assignments are there?". Obviously $\phi \in$ SAT iff $\#\text{SAT}(\phi) \neq 0$.

Some results:

$$\text{FP} \subseteq \#\text{P} \subseteq \text{FPSPACE}, \quad \text{P}^{\text{PP}} = \text{P}^{\#\text{P}}, \quad \text{FL} \subseteq \#\text{L} \subseteq \text{FNC}^2.$$

Theorem 6.1.3 (Toda)**.** $\text{PH} \subseteq \text{P}^{\#\text{P}}$.

6.2 Approximation algorithms

In this section we consider optimization problems. These are search problems of a feasible solution that maximizes or minimizes an objective function.

An optimization problem is: (I, S, v, goal).

- I: problem instances;
- S: a function that maps instances to feasible solutions;
- v: the objective function maps feasible solutions to positive integers
- goal: min or max, for minimization or maximization of the objective function, respectively.

The value of the objective function at the optimal solution for input x is denoted by $\mathrm{OPT}(x)$ and is equal to $\mathrm{goal}\{v(y) \mid y \in S(x)\}$.

For each optimization problem, we define the underlying decision problem:

Given, in addition to x, a bound k.
Question: is $\mathrm{OPT}(x) \geq k$?
(for a maximization problem – similarily for a minimization problem)

E.g., in MAX-CLIQUE the instance is a graph x, the feasible solutions are all complete subgraphs of x (cliques), the objective function is the number of nodes in the clique and goal = max. The underlying decision problem is CLIQUE.

Optimization problem classes:

Definition 6.2.1. NPO: the class of optimization problems for which the underlying decision problem is in NP (with the condition that there are feasible solutions for every instance).

Definition 6.2.2. PO: the class of optimization problems for which the underlying decision problems is in P.

Many optimization problems are NP-hard, so approximation algorithms are useful in this case.

Definition 6.2.3. A polynomial time algorithm M is ρ-approximation for a maximization problem if for every $x \in I$ it gives a solution $M(x) \in S(x)$ such that:

$$\frac{v(M(x))}{\mathrm{OPT}(x)} \leq \rho.$$

Similarily, for a ρ-approximation algorithm for a minimization problem.

Some important NPO subclasses:

- poly-APX: problems for which there exists a $p(n)$-approximation algorithm for some polynomial p (where n is the input size: $n = |x|$).

- log-APX: problems for which there exists a $\log n$-approximation algorithm (where n is the input size: $n = |x|$).
- APX: problems for which there exists a ρ-approximation algorithm for some constant $\rho > 0$.
- PTAS: problems for which there exists a *polynomial time approximation scheme*, i.e., a $(1+\varepsilon)$-approximation algorithm *for any* constant $\varepsilon > 0$.
- FPTAS: problems for which there exists a *fully polynomial time approximation scheme*, i.e., a $(1+\varepsilon)$-approximation algorithm *for any* constant $\varepsilon > 0$, where, additionally, the time needed is also polynomial w.r.t. $1/\varepsilon$.

The inclusion between the above classes is shown in figure 6.1.

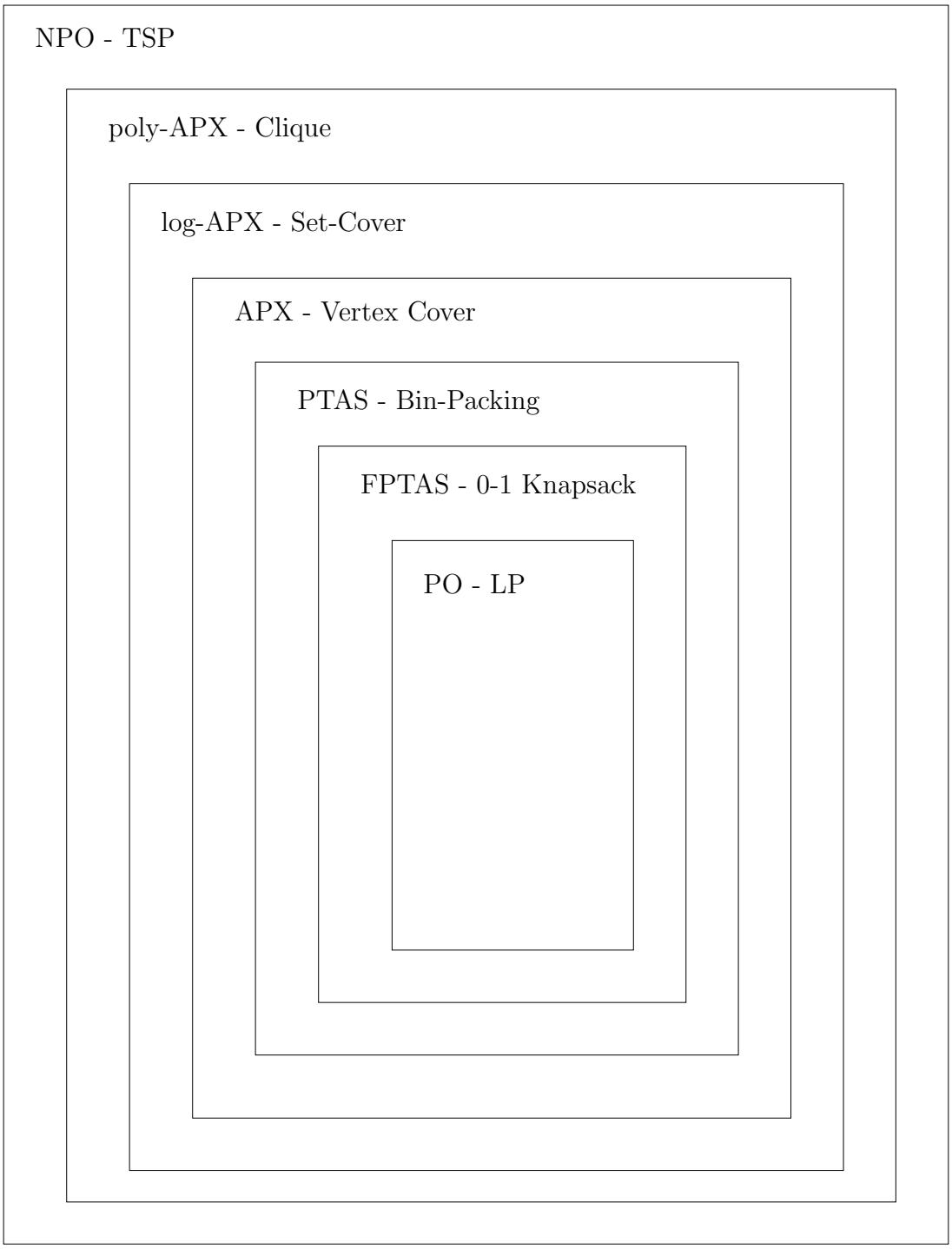

Figure 6.1: Optimization classes