Definitions

Numerical Problems

• The problem: Given a purse with a set of coins, and given an amount of money to pay, can you pay the amount precisely?

  Formally: An instance $\langle A, S \rangle$ of \textsc{SubsetSum} is a set of positive integers $A = \{a_1, \ldots, a_n\}$ and a positive integer $S$. $\langle A, S \rangle$ is a positive instance, iff there is a subset $A' \subseteq A$ such that $\sum_{a \in A'} a = S$.\(^1\)

• A special case of \textsc{SubsetSum} is \textsc{Partition}– given a set of coins, can you divide it into two sets of equal value?

  Formally: A \textsc{Partition}-instance $\langle A \rangle$ is a positive one, iff there is a subset $A'$ such that $\sum_{a \in A'} a = \sum_{a \in A - A'} a$.

• The Problem: Given a set of items (each with a value and a weight), can you find a subset of maximum value fitting your knapsack?

  Formally: An instance $\langle A, w, v, W, V \rangle$ of \textsc{Knapsack} comes with a set of items $A = \{a_1, \ldots, a_n\}$, a weight function $w : A \rightarrow \mathbb{N}$, a value function $v : A \rightarrow \mathbb{N}$, a positive capacity $W$, and the minimal value $V$ to be packed into the knapsack.

  The task is to find a subset $A' \subseteq A$ such that $\sum_{a \in A'} w(a) \leq W$ with $\sum_{a \in A'} v(a) \geq V$.

\(^1\)Note that this formulation does not allow to have more than one item with the same weight. Whether multiple items of the same weight are allowed or not is unimportant with respect to the complexity of the problem.
- The Problem: Given a set of items (each with a certain weight), how many knapsacks do you need to carry them?

Formally: An instance \( \langle A, C, B \rangle \) of \textsc{BinPacking} is a set of positive integers \( A = \{a_1, \ldots, a_n\} \) and two positive integer \( C \) and \( B \). Given an instance \( \langle A, C, B \rangle \), the task is to find a partition \( \langle B_1, \ldots, B_k \rangle \) of \( A \) with
  
  - \( k \leq B \)
  - \( \sum_{a \in B_i} a \leq C \) for \( 1 \leq i \leq k \)
  - \( \bigcup_{i=1}^k B_i = A \).

Graph Problems

- A \textsc{VertexCover}-instance is consists of an undirected graph \( \langle V, E \rangle \) and a positive bound \( K \). The problem is to find a subset \( V' \subseteq V \) with \( |V'| \leq K \) such that \( \{u, v\} \cap V' \neq \emptyset \) for all \( \langle u, v \rangle \in E \).

- \textsc{HyperGraphVertexCover} is a generalization of \textsc{VertexCover}. Hypergraphs allow more than two vertices per edge, i.e., in a hypergraph \( \langle V, H \rangle \) an hyperedge \( h \in H \) is subset of \( V \). \textsc{HyperGraphVertexCover} asks for a subset \( V' \subseteq V \) with \( |V'| \leq K \) such that for all \( h \in H \) we have a \( v \in V' \) with \( v \in h \).

- \textsc{DominatingSet} has directed graphs \( \langle V, E \rangle \) together with a bound \( K \) as instances. The task is to find a subset \( D \subseteq V \) with \( |D| \leq K \) such that for each \( v \in V - D \) there is a \( u \in D \) with \( \langle u, v \rangle \in E \).

Set Problems

- \textsc{SetCover} has instances of the form \( \langle C, U, K \rangle \) where \( c \in C \) are subsets of the universe \( U \) and \( K \) is a positive integer. The task is to find a subset \( C' \subseteq C \) with \( |C'| \leq K \) such that \( \bigcup_{c \in C'} c = U \).

Sat-Variants

- \textsc{2Sat} is the set of satisfiable CNF-expressions with at most 2 literals per clause, i.e., it is the problem of determining whether a 2-CNF formula is satisfiable.

- \textsc{HornSat} is another restriction of \textsc{Sat}. An instance of \textsc{HornSat} contains only clauses which contain at most one positive literal (\( x \lor \neg y \lor \neg z \) is a Horn-clause, but \( x \lor y \lor \neg z \) is not a Horn clause). \textsc{HornSat} is the problem of deciding whether such an instance is satisfiable or not.
Problem 1 – Membership in NP

Show that the problems of the last section are in NP.

1. [2] SubsetSum
2. [2] Partition
5. [2] VertexCover
6. [2] HyperGraphVertexCover
7. [2] DominatingSet
8. [2] SetCover
9. [2] 2Sat

Problem 2 – Example Instances for problems in NP

Give a non-trivial yes-instance and a non-trivial no-instance for each example problem. E.g., in the case of Partition, a yes-instance is a set of positive integers which can be partitioned into two disjoint subsets such that the sum of the elements of the two subsets is equal. A no-instance of Partition is a set of positive integers where no such partition exists.

1. [1] SubsetSum
2. [1] Partition
4. [1] BinPacking
5. [1] VertexCover
6. [1] HyperGraphVertexCover
7. [1] DominatingSet
8. [1] SetCover
9. [1] 2Sat
10. [1] HornSat
Problem 3 – First Reduction

In the lectures, the problems 3Sat and 3Coloring were introduced. We will establish their equivalence with respect to their computational complexity.

1. [5] Give a reduction from 3Sat to 3Coloring in terms of an algorithm (pseudo-code).
2. [5] Give the converse reduction from 3Coloring to 3Sat, again in terms of an algorithm.
3. [3] Illustrate these reductions with two non-trivial examples – a yes-instance and a no-instance. Show the original instance of 3Sat, the result of the first reduction, and use the this result as input for the second reduction and show the corresponding result.

Problem 4 – Numerical NP-complete problems

1. [10] Reduce one of the NP-complete problems of the lecture (CircuitSat, SAT, 3Sat, 3Coloring, ...) to SubsetSum.
   (Hint: 3Sat is probably the easiest one). Give a good example for the reduction.
2. [5] Reduce SubsetSum to Partition. Continue the example of the preceding item.
3. [3] Show that Partition is a special case of Knapsack.
4. [3] Show that Partition is a special case of BinPacking.
5. [10] Give an algorithm which solves a Knapsack-instance \( \langle A, w, v, W, V \rangle \) in time \( \text{poly}(v_{\text{max}}|A|) \) where \( v_{\text{max}} = \max\{v(a) : a \in A\} \). Why is this not a polynomial time algorithm?

Problem 5 – NP-complete Graph Problems

1. [6] Reduce 3Sat to VertexCover. Give a good example to demonstrate your reduction.
   Hint: This proof is easiest done with a gadget construction: You will need a guess gadget which models the possible truth assignments and you will need a check gadget which checks that the chosen assignment satisfies the clause set.
2. [3] Show that HyperGraphVertexCover and SetCover are different formulations of the same problem.
3. [8] HyperGraphVertexCover is generalization of VertexCover, thus the reduction from VertexCover to HyperGraphVertexCover is trivial. Give the converse reduction.
   Hint: This proof can be established with a local-replacement construction, i.e., you can replace each hyperedge with some ordinary graph. Give an instructive example to illustrate your reduction.
4. [3] Reduce VERTEXCOVER to DOMINATINGSET. 
   Hint: Again, a local-replacement will solve the problem easily.

**Problem 6 – 2Sat and HornSat are in P**

We proved that 3Sat is \( \text{NP} \)-complete. Restricting this problem quickly leads to feasible instances:

1. [5] Prove that 2Sat is solvable in polynomial time.

2. [15] Prove that HORNSAT is solvable in polynomial time.

**Problem 7 – Self-Reducibility ([6])**

Assume that you have a library with one API-function \textsc{decideSat}: It takes an arbitrary SAT-instance \( \phi \) as argument and returns true iff \( \phi \) has a satisfying assignment – magically within constant time.

Describe a small polynomial time algorithm \textsc{findAssignment} which takes a CNF-formula \( \phi \) as input and uses \textsc{decideSat} to compute and output a concrete satisfying truth assignment (naturally, if \( \phi \) is indeed satisfiable).

This property of SAT is called self-reducibility. It is useful as a building block in some proofs.