Approximation Algorithms - inofficial lecture notes

April 26, 2016

Contents

Chapter 1

NP-Completeness and Problems

1.1 NP-Completeness

Definition 1.1 (Basics). An **alphabeth** is a finite set of symbols. A **string** is a finite sequence of symbols. The **length of a string** is the number of symbols in the string, denoted by $|S|$.
If Σ is an alphabeth and $n \in \mathbb{N}$, then Σ^n denotes the. set of all strings with length n and symbols in Σ. Moreover define

$$\Sigma^* := \bigcup_{n=0}^{\infty} \Sigma^n.$$

Definition 1.2 (Decision Problem). A **Decision Problem** is a problem allowing the answer yes (1) or false (0). This corresponds to subsets of Σ^*, the so called **languages**. Let $X \subseteq \{0,1\}^*$ be a decision problem. Algorithm A **decides X** if we have: A return value $A(s) \in \{\text{“yes”}, \text{“no”}\}$ s.t.

$$A(s) = \text{“yes”} \quad \Longleftrightarrow \quad s \in X.$$

A has **polynomial runtime** if there exists some polynomial p s.t. the number of steps (i.e. the number of head moves of a touring machine) is bounded by $O(p(|s|))$ for input s.

Problems can be grouped according to their difficulty:

P All decision problems for which a polynomial time algorithm exists

NP All problems where the solution can be verified in polynomial time

Let X be a decision problem. We have that $X \in NP$ if there exists a polynomial time

algorithm $A : \{0,1\}^* \times \{0,1\}^* \to \{\text{“yes”}, \text{“no”}\}$ and a polynomial p s.t. $s \in X \Leftrightarrow \exists t \in \{0,1\}^*$ s.t. $|t| \leq p(|s|)$ and $A(s,t) = \text{“yes”}$.

1.2 Problems of complexity NP

In this section the most standard NP-complete problems are presented.

1.2.1 Reducibility

Let X and Y be decision problems. We say that Y is **polynomial time reducible to** X (or Y polynomially reduces to X) if there exists a function

$$f : \{0,1\}^* \to \{0,1\}^x$$

computable in polynomial time s.t. $f(s) \in X \Leftrightarrow s \in Y$.

Theorem 1.3
If Y is polynomial time reducible to X and there exists a polynomial time algorithm for X then there exists a polynomial time algorithm for Y.

Definition 1.4. A problem X is called **NP-hard**, if all problems in NP polynomially reduce to X. It is called **NP-complete**, if it is NP-hard and in NP.

Theorem 1.5
If there exists a polynomial time algorithm for an NP-complete problem, then P=NP

1.2.2 Satisfiability (SAT)

Input: A boolean formula F in conjunctive normal form

Question: Is F satisfiable?

Theorem 1.6 (Cook)
Satisfiability is NP-complete.

Input: A boolean formula F in conjunctive normal form where each clause contains at most k literals

Question: Is F satisfiable?

Theorem 1.7
k-SAT is NP-complete for $k \geq 3$

Remark 1.8. 2-SAT is in P.

1.2.3 Directed Hamiltonian Cycle Problem

Input: A digraph G

Question: Does G contain a cycle of length $|V(G)|$?

Theorem 1.9 (Karp (Reductability among comb. problems))
Directed Hamiltonian Cycle is NP-complete.

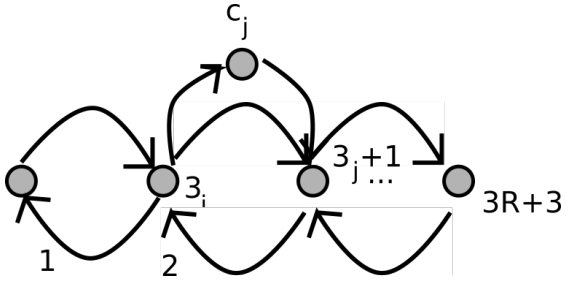

Figure 1.1: The path used for the proof of Theorem 1.11

Proof. The problem is obviously in NP. Reduce 3-SAT to Directed Hamiltonian Cycles as follows:

Let $F = C_1 \wedge C_2 \wedge ... \wedge C_k$ be a 3-SAT formula in variables $x_1, ..., x_n$. Construct G as follows: Path P_i for $i = 1, ..., n$ on $3R + 3$ vertices (see Fig. 1.1). Add vertices s and t and connect the paths as in Fig. 1.2. For each clause C_j add a vertex c_j. Connect c_j with vertex $3j + 1$ in P_i and connect vertex $3j$ in P_i with c_j if x_i appears in C_j. For $\bar{x}_i$: Reverse direction. F satisfies: Fix some satisfying truth assignment. Traverse P_i rom left to right $\Leftrightarrow$ x_i is true. Choose for each C_j a true literal (see Fig. 1.3).

Assume: A directed Hamiltonian Circuit exists.

Observation: If the circuit enters a vertex c_j from path P_i then the next edge will go back to P_i (see Fig. 1.4) $\Rightarrow$ We can remove all the c_j vertices from the hamiltonian circuit by replacing the two edges by one edge from some $P_i \Rightarrow$ Use direction of the P_is as truth values. ∎

Remark 1.10. The same statement holds for Hamiltonian paths.

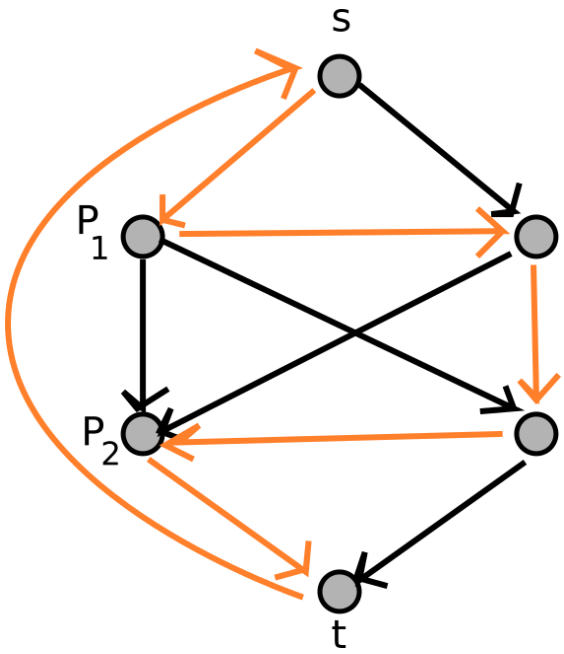

Figure 1.2: Construction used for the proof of Theorem 1.11

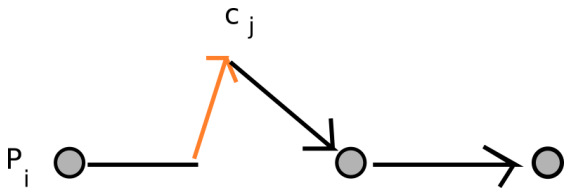

Figure 1.3: Another idea needed for proof of Theorem 1.11

1.2.4 Hamiltonian Cycle

Input: An undirected graph $G = (V, E)$

Question: Does G contain a cycle of length $|V(G)|$?

Theorem 1.11
Hamiltonian Cycle is NP-complete.

1.2.5 Vertex Cover

Input: A Graph $G = (V, E), R \in \mathbb{N}$

Question: Does G contain an $S \subseteq V$ s.t. $|S| \leq R$ s.t. each edge in E has at least one

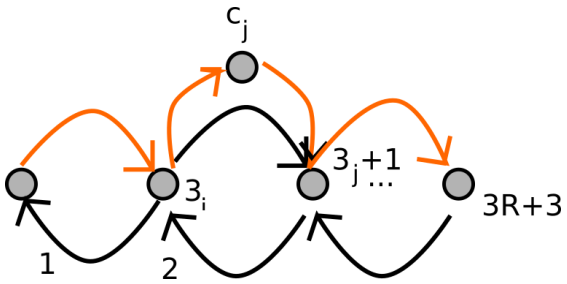

Figure 1.4: Another idea needed for proof of Theorem 1.11

endpoint in S?

Theorem 1.12 (Karp 1972)
The Problem of Vertex Cover is NP-Complete.

PROOF. Obviously the problem is in NP. Reduce 3-SAT to Vertex Cover.

Let $F = C_1 \wedge ... \wedge C_q$ be a 3-SAT formula. W.l.o.g. each clause contains exactly 3 literals. Construct G as follows: $3q$ vertices

Edges:

1) Connect the three vertices corresponding to the 3 literals of the same clause.

2) Connect the vertices a, b if they correspond to negations of each other.

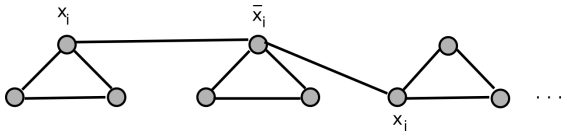

Figure 1.5: The constructed Graph for the proof of Theorem 1.12

Claim: G has a vertex cover of size $2q \Leftrightarrow F$ is satisfiable.

F is satisfiable $\rightarrow$ Fix some truth assignment. Choose some true literal from each clause. Take all non-chosen vertices as vertex cover. This is really a vertex cover, because we cannot have that x_i and $\bar{x}_i$ are true for some i.
G has a vertex cover of size $2q \Rightarrow$ vertex cover contains exactly 2 vertices from each triangle (clause). Set all uncovered vertices to true; it is well defined as $\{x_i, \bar{x}_i\}$ is an edge for all i and ust be covered. ∎

title

Stable Set **Input:** A graph $G = (V, E), R \in \mathbb{N}$

Question: Is there an $S \subseteq V$ with $|S| = R$ s.t. no edge in E connects two vertices in S?

Theorem 1.13 (Karp)
Stable Set is NP-Complete.

PROOF. Observation: $S \subseteq V$ is a stable set $\Leftrightarrow V \setminus S$ is a vertex cover. Moreover $S \subseteq V$ is a stable set of size $n - k \Leftrightarrow V \setminus S$ is a vertex cover of size k.

title

Clique Problem **Input:** A graph $G = (V, E), R \in \mathbb{N}$

Question: Is there an $S \subseteq V, |S| = R$ s.t. all possible edges between vertices in S belong to E.

Theorem 1.14 (Karp 1972)
The Clique Problem is NP-Complete.

PROOF. It is obviously in NP. Moreover, S is a stable set in $G \Leftrightarrow S$ is a clique in $\bar{G}$. ∎

title

Set Cover **Input:** A family $S_1, ..., S_m$ of subsets of a finite set S, $R \in \mathbb{N}$

Question: Is there a subfamily $S_{i_1}, ..., S_{i_R}$ s.t. $\bigcup_{j=1}^{R} S_{i_j} = S$?

Theorem 1.15 (Karp 1972)
Set cover is NP-Complete.

PROOF. It is obviously in NP. Reduce Vertex-Cover to Set Cover as follows:

Let $G = (V, E), k \in \mathbb{N}$ be a vertex cover instance. Set $S \coloneqq E$ and $S_x \coloneqq \delta(x)$ for $x \in V(G)$. ∎

1.2.6 Integer Linear Inequalities

Input: A matrix $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$

Question: Is there a vector $x \in \mathbb{Z}^n$ s.t. $Ax = b$?

Theorem 1.16 (Karp 1972)
Integer Linear Inequalities is NP-Complete

PROOF. The problem is in NP. Reduce 3-SAT to Integer Linear Inequalities:

Let $F = C_1 \wedge ... \wedge C_q$, in the variables $x_1, ..., x_n$. We use the following inequalities:

$$\left.\begin{aligned} -x_i - \bar{x}_i &\leq -1 \\ x_i + \bar{x}_i &\leq 1 \\ -x_i &\leq 0 \\ -\bar{x}_i &\leq 0 \end{aligned}\right\} \Rightarrow \begin{aligned} & x_i - 0 \wedge \bar{x}_i = 1 \\ \text{or } & x_i = 1 \wedge \bar{x}_i = 0 \end{aligned}$$

■

1.2.7 3-Dimensional Matching (3DM)

Input: Disjoint sets X, Y, Z each of size n and $T \subseteq X \times Y \times Z$ a set of hyperedges

Question: Is there an $M \subset T$ with $|M| = n$ s.t. $\forall x \in X \times Y \times Z$ there exists exactly one $m \in M$ containing a?

Theorem 1.17 (Karp 1972)
3DM is NP-Complete.

PROOF. Obviously 3DM is in NP. Reduce 3-SAT to 3DM as follows:

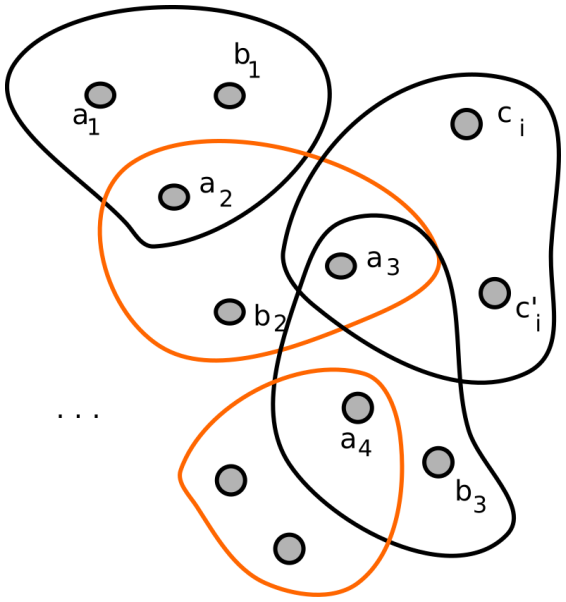

Figure 1.6: Idea to be used in the proof of Theorem 1.17

Let $F = C_1 \wedge ... \wedge C_k$ be a 3-SAT formula in the variables $x_1, ..., x_n$. For each variable x take vertices $a_1^x, ..., x_{2k}^x, b_1^x, ..., b_{2k}^x$. Then we add triples $(a_i^x, a_{i+1}^x, b_i^x)$ (indices taken "mod" 2k), see also Fig. 1.6.

A triple $(a_i^x, a_{i+1}^x, b_i^x)$ is called even if i is even. Otherwise it is called odd.

b_i^x is called the tip of $(a_i^x, a_{i+1}^x, b_i^x)$.

Even triple $\equiv$ x is set to false $\equiv$ *oddtipsarenotcovered*.

Analogeously Odd triple $\equiv$ x is set to true $\equiv$ even tips are not covered.

C_i is a clause $\Rightarrow$ add vertices c_i and c_i' and t riples $(c_i, c_i', b_{2_i}^x)$ if C_i contains x, and (c_i, c_i', b_{2i-1}^x) if C_i contains $\bar{x}$. Using this we cover k tips from the uncovered kn tips. Thus $kn - k$ still remain uncovered. In order to also cover thos, we add cleanup triples (q_i, q_i', b_j^x) for $i = 1, ..., (n-1)k$ and $j = 1, ..., k$ for all x.

F satisfiable $\Rightarrow$ 3DM exists!

If 3DM exists $\Rightarrow$ all even or odd triples are chosen for some variable $x \Rightarrow$ set x to true iff odd triples are chosen. We know that C_j is covered, i.e. there is (c_j, c_j', b_{2i}^x) exists $\Rightarrow$ $x \in C_j$, so x is true $\Rightarrow$ C_j is true. ∎

title

Subset Sum **Input:** $a_1, ..., a_n \in \mathbb{N}, K \in \mathbb{N}$

Question: Is there an $S \subseteq \{1, ..., n\}$ s.t. $\sum_{i \in S} a_i = K$?

Theorem 1.18 (Karp 1972)
Subset Sum is NP-Complete.

Proof. Obviously Subset Sum is in NP. Reduce 3DM to Subset Sum:

Take an instance of 3DM, so let X, Y, Z sets with $|X| = |Y| = |Z| = n$ n and $T \subset X \times Y \times Z$ with $|T| = m$. Each $t \in T$ can be seen as a $0-1$-vector of length $3n$ with exactly 3 entries equal to 1, corresponding to the three elements covered by t. Think of this vector being a number with $3n$ digits with base $m + 1$. Let K be the number corresponding to the all-1-vector. Let a_t be the number corresponding to $t \in T$. Then 3DM has a solution $\Leftrightarrow \exists S \subset T$ s.t. $\sum_{s \in S} a_s = K$. ∎

1.2.8 Partition

Input: $a_1, ..., a_n \in \mathbb{N}$

Question: Is there an $S \subseteq \{1, ..., n\}$ s.t. $\sum_{i \in S} a_i = \sum_{i \notin S} a_i$?

Theorem 1.19 (Karp 1972)
Partition is NP-Complete.

PROOF. Obviously the Partition-Problem is in NP. Reduce Subset Sum to Partition:

Let $a_1, ..., a_n \in \mathbb{N}, k \in \mathbb{N}$ be an instance of Subset Sum. If $K = \frac{1}{2}\sum_{i=1}^{n} a_i$. Otherwise w.l.o.g $K \geq \frac{1}{2}\sum_{i=1}^{n} a_i$. Then one can compute $a_{n+1} \in \mathbb{N}$ s.t.

$$K\frac{1}{2}\sum_{i=1}^{n+1} a_i.$$

Then we have:

$$\sum_{i\in S} a_i = K \quad \Longleftrightarrow \quad \sum_{i\in\{1,...,n+1\}\setminus S} a_i = \sum_{i\in S} a_i,$$

since

$$\sum_{i\in\{1,...,n+1\}\setminus S} a_i + \sum_{i\in S} a_i = 2K$$

and we are done. ∎

1.2.9 Max Cut

Input: A Graph $G = (V, E)$, $w : E \to \mathbb{N}, K \in \mathbb{N}$

Question: Is there an $S \subset V$ s.t. $w(\delta(S)) \geq K$?

Theorem 1.20 (Karp 1972)
Max Cut is NP-Complete.

PROOF. Obviously the problem is in NP. Reduce Partition to Max Cut as follows:

Let $a_1, ..., a_n \in \mathbb{N}$ an instance to Partition. Let G be a complete graph on n vertices. Assume the vertices have labels $\{1, ..., n\}$. Define

$$w(\{i, j\}) \coloneqq a_i \cdot a_j, \quad i \neq j.$$

Now define

$$K \coloneqq \frac{1}{4}\left(\sum_{i=1}^{n} a_i\right)^2.$$

For $S \subseteq \{1, ..., n\}$ we have then

$$w(\delta(S)) = \sum_{i\in S} a_i \cdot \sum_{j\notin S} a_j \overset{\text{AGM}}{\leq} \frac{1}{4}\left(\sum_{i=1}^{n} a_i\right)^2$$

and equality holds iff

$$\sum_{i\in S} a_i = \sum_{i\notin S} a_i$$

holds. ∎

1.2.10 Component Grouping

Input: A Graph $G, K \in \mathbb{N}$

Question: Is there a set of connected components in G that contains exactly K vertices?

Theorem 1.21
Component Grouping is in P, but NOT NP nor NP-Complete.

Proof. Idea: Dynamic Programming. As follows:

Define a function

$$f(i,k) \coloneqq \max_{S \subseteq \{1,\dots,i\}, \sum_{j\in S} a_j \leq K} \sum_{j \in S} a_j,$$

using a_j to be the size of the j-th component in our graph. If we can evaluate the function for all possible arguments we are done, as the problem just requieres the solution of $f(n,K) = K$. We get the needed function as follows: First define

$$f(0,k) \coloneqq 0$$

and then

$$f(i,k) \coloneqq \begin{cases} \max\{f(i-1,k), a_i + f(i-1,k-a_i)\}, & a_i \leq k \\ f(i-1,k), & \text{otw.} \end{cases}$$

for $i \in \{1,\dots,n\}$ and $k \leq n \Rightarrow O(n^2)$ algorithm. ∎

1.2.11 k-Colorability

Input: A Graph G

Question: Is there a function $f : V(G) \to \{1,\dots,k\}$ s.t. $f(x) \neq f(y)$ if $\{x,y\} \in E(G)$ (connected nodes need to have different color, f is the colouring function)
Remark 1.22. Karp proved the Colorability Problem back in 1972 to be NP-Complete. Moreover we know already, that 2-Colorability is in P.

Theorem 1.23 (Garey, Johnson, Stockmeyer, 1976)
3-*Colorability is NP-Complete.*

Proof. Obviously the problem is in NP. Using the graphs showing in Fig. 1.7 we can use a reduction from 3-SAT:

Construct such a graph for each clause $C_1,\dots,C_m$. Identify vertices that correspond to the same literal. As seen in Fig. 1.8 add edges between x_i and $\overline{x_i}$ for all x_i. Moreover call the color assigned to vertex B "black". Connect B and R to all z- and z'-vertices. This was the proof for the 3-coloring problem. The k-Coloring Problem can be reduced completely to this problem! ∎

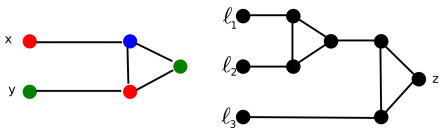

Figure 1.7: Graph used for the proof of Theorem 1.23; As seen on the left, the color for x and y are uniquely defined by the color of z

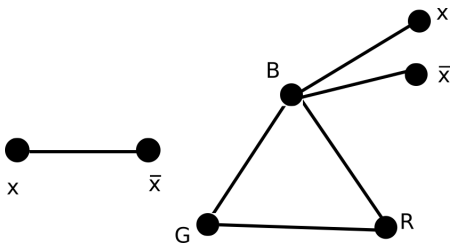

Figure 1.8: How the graph is constructed for the proof of Theorem 1.23

The **complement** of a decision problem $P \subseteq \{0,1\}^*$ is the decision problem $\{0,1\}^* \setminus P$.

CO-NP is the set of all decision problems s.t. their complement is in NP.

1.2.12 Prime

Input: A number $n \in \mathbb{N}$

Question: Is n prime?

Theorem 1.24 (Pratt 1975)
Prime is in NP.

Remark 1.25. Moreover Prime is in CO-NP∩ NP. This is a good characterization. Moreover Prime is in P due to Agraval, kayal, Saxena, 2004.

1.3 Discrete Optimization Problems

Definition 1.26. A **Discrete Optimization Problem** is a four-tuple $(X, S, c, goal)$ where

- $X \subseteq \{0,1\}^*$ can be decided in polynomial time. The elements of X are called **instances**
- $S : X \to 2^{\{0,1\}^*}$ where $\emptyset \neq S(x) \subseteq \{0,1\}^*$ is the **set of solutions** of an instance x s.t. there exists a polynomial p with $|y| \leq p(|x|)$ for all $y \in S(x)$ and all $x \in X$
- $c : \{(x,y) |\, x \in X, y \in S(x)\} \to \mathbb{Z}$ is a function that is computable in polynomial time
- $goal \in \{\min, \max\}$
- $OPT(x)$ is the value $goal\{c(x,y) |\, y \in S(x)\}$
- An **optimum solution** for an instance x is a $y \in S(x)$ with $c(x,y) = OPT(x)$
- An **Algorithm for an optimization problem** $(X, S, c, goal)$ is an algorithm that computes for each $x \in X$ a $y \in S(x)$

Definition 1.27. We denote the output of an algorithm A by $A(x) \coloneqq c(x,y)$.

If $A(x) = OPT(X)$ for all $x \in X$ then A is called an **exact algorithm**.

We have just defined NP and NP-Completeness for decision problems. What is now an NP-Hard Optimization Problem? We want to extend our definition.

Definition 1.28. An **Underlying decision problem** for an optimization problem $(X, S, c, goal)$ is defined as

$$\{(I,k) |\, I \in X, k \in \mathbb{Z}, \text{ s.t. } \exists s \in S(I) \text{ with } c(I,S) \leq / \geq k\}$$

If the sign is $\leq$ or $\geq$ depends on whether we want to minimize or maximize.

Definition 1.29. A discrete optimization problem is NP-hard if its underlying decision problem is NP-hard.

Some NP-hard optimization problems (decision problems) can be solved in polynomial time, if the numbers appearing in the input are "not to large". Let X be a decision problem. Define a function

$$Max : X \to \mathbb{N}$$

that assigns the magnitude of the largest number appearing in an instance to the instance.

Assumption 1.30
Max is "reasonable".

Remark 1.31. In the above assumption, what should "reasonable" stand for? Analogeously to the *length of an instance* there is no clear definition of it. One would need to talk about reasonability of the corresponding encoding for a problem (e.g. store a graph as adjacency-matrix, etc.). Implicitly we will assume from now on, that all problems we are dealing with come coupled with a corresponding length- and Max-function.

Definition 1.32. A **Pseudo-polynomial algorithm** is an algorithm, whose runtime may depend polynomially in $|x|$ and $Max(x)$ for an instance x. If $Max(x)$ is polynomially in $|x|$, then polynomially and pseudo-polynomial algorithms coincide. If $Max(x)$ is not polynomially in $|x|$, then the problem is called a **number problem**.

Theorem 1.33
There exists a pseudo-polynomial algorithm for Subset Sum, polynomial in $|x|, Max(x)$.

Proof. Take algorithm for Component Grouping $O(n, K)$. ∎

Corollary 1.34
There exists a pseudo-polynomial algorithm for Partition.

Proof. Use the algorithm for Subset Sum with $K := \frac{1}{2}\sum_{i=1}^{n} a_i$. ∎

Definition 1.35. An NP-complete (NP-hard) problem is called **strongly** NP-complete (**strongly** NP-hard), if there exists some polynomial p s.t. the problem stays NP-complete (NP-hard), if restricted to instances with $Max(x) \leq p(|x|)$

Theorem 1.36 (Garey, Johnson 1978)
Unless P=NP, there does not exist an exact pseudo-polynomial algorithm for any strongly NP-hard problem.

Corollary 1.37
Integer Linear Inequalities is strongly NP-complete.

Proof. See proof of Theorem 1.16: All numbers in the reduction were in $\{-1, 0, 1\}$. ∎

1.3.1 Simple Max Cut

Input: A graph $G = (V, E), K \in \mathbb{N}$

Question: Is there an $S \subseteq V$ s.t. $|\delta(S)| \geq K$?

Theorem 1.38 (Garey, Johnson, Stockmeyer 1976)
Simple Max Cut is NP-Complete.

Corollary 1.39
Max Cut is strongly NP-complete.

Proof (Proof of Theorem 2.2). Obviously the problem is in NP. Reduction from Stable Set:

Let $G = (V, E), K$ be an instance of stable set. Construct a graph $G' = (V', E')$ as follows:

$$V' \quad := \quad V \cup \{x\} \cup \{\{e, i\} | \, e \in E(G), i = 1, 2\}$$

Definition of E' by picture in Fig. 1.9 **Claim:** G has a stable set of size $K \Leftrightarrow G'$ has a

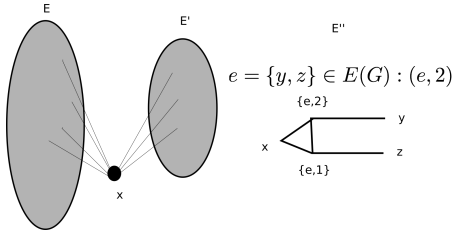

Figure 1.9: The way E' is defined shown in a picture

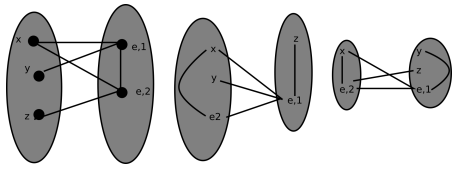

Figure 1.10: Cuts created. In each case we have 5 additional vertices

cut with $K + 4|E|$ edges.

" $\Rightarrow$" S is a stable set of size K. Add the vertices $(e, 1), (e, 2)$ appropriately, s.t. 4 edges of the edge gadget are cut. $\Rightarrow 4|E| + K$ edges.

" $\Leftarrow$" For each edge-gadget at most 4 edges are in the cut $\Rightarrow$ at least K edges of the form $\{x, v\}$ are in the cut

$$S_C := \{v \in V | \, \{x, v\} \in \text{ cut}\}, \quad |S_C| = K + \ell, \quad \ell \geq 0$$

so $\leq \ell$ edges for which only 3 edges of the edge gadget are in the cut. For each such edge (3 edges in the edge gadget) remove one of the endpoints from S_C and call the

remaining vertices S. Then S is a stable set and

$$|S| \geq |S_C| - \ell = K + \ell - \ell = K.$$

■

Conclusion: We have seen that for some of the "easy" problems, a pseudo-polynomial time algorithm exists.

Chapter 2

Approximation Algorithms

In this example we will give some example of Approximation Algorithms.

Definition 2.1. An **Approximation Algorithm** A for an optimization problem P is a *polynomial time* algorithm for P. A has **approximation ratio** c, if (for $c \in \mathbb{R}$) we have

$$\begin{aligned} A(I) &\leq c \cdot OPT(I) \quad \text{if } P \text{ is a minimization problem} \\ A(I) &\geq c \cdot OPT(I) \quad \text{if } P \text{ is a maximization problem} \end{aligned}$$

If P is a minimization problem we assume $1 \leq c \leq \infty$ and if P is a maximization problem then $0 \leq c \leq 1$ holds.

2.1 Minimum Weight Set Cover Problem

Input: A family $S_1, ..., S_m$ of subsets of a finite set U with $\bigcup_{i=1}^{n} S_i = U$, $c : \{S_1, ..., S_m\} \to \mathbb{R}_+$

Output: A subfamily $S_{i_1}, ..., S_{i_k}$ s.t. $\bigcup_{j=1}^{k} S_{i_j} = U$ s.t.

$$\sum_{j=1}^{k} c(S_{i,j})$$

is minimal.

Greedy Weighted Set Cover Algorithm

Input: $S_1, ..., S_m, U, \bigcup S_i = U, c : \{S_1, ..., S_m\} \to \mathbb{R}_+$

Output: Optimum subfamily E of the S_i

- $C \coloneqq \emptyset$, $E \coloneqq \emptyset$
- while $C \neq U$
- TAB choose $S \in \{S_1, ..., S_m\}$ s.t. $\frac{c(S)}{|S \setminus C|}$ is minimum
- TAB $C \coloneqq C \cup S$
- TAB $E \coloneqq E \cup S$

Theorem 2.2
The greedy weighted set algorithm has approximation ratio

$$H_n \coloneqq \sum_{i=1}^{n} \frac{1}{i}, \quad \text{if } n \coloneqq |U|.$$

PROOF. W.l.o.g assume $E = \{S_1, ..., S_k\}$ (i.e. we order our sets in a way s.t. the algorithm returns the first k of them). For each $x \in U$ assign cost

$$\tilde{c}(x) \coloneqq \frac{c(S_i)}{|S_i \setminus \{S_1 \cup ... \cup S_{i-1}\}|},$$

where i is chosen minimum s.t. $x \in S_i$.

Claim: For $S \in \{S_1, ..., S_m\}$ we have

$$\sum_{x \in S} \tilde{c}(x) \leq c(S) \cdot H_{|S|}.$$

If we prove this claim we are done: Let E^* be an optimum solution. Then

$$\begin{aligned} \sum_{S \in E} c(S) &= \sum_{x \in U} \tilde{c}(x) \\ &\leq \sum_{S \in E^*} \sum_{x \in S} \tilde{c}(x) \\ &\overset{\text{claim}}{\leq} \sum_{S \in E^*} c(S) \cdot H_{|S|} \leq H_n \cdot \sum_{S \in E^*} c(S). \end{aligned}$$

Proof of the claim: Let $S = \{x_1, ..., x+p\} \in \{S_1, ..., S_m\}$ s.t. x_i is covered not earlier than x_j, if $i \geq j$. Consider the step where the greedy algorithm covers x_i for the first time, $1 \leq i \leq p$. The greedy algorithm could have chosen S. We know for sure that

$$\tilde{c}(x_i) \leq \frac{c(S)}{|S| - (i-1)}.$$

Therefore

$$\begin{aligned}\sum_{x\in S}\tilde{c}(x) &\leq \sum_{i=1}^{|S|}\frac{c(S)}{|S|-(i-1)}\\ &= c(S)\sum_{i=1}^{|S|}\frac{1}{i}\\ &= c(S)H_{|S|},\end{aligned}$$

and this is the claim. ∎

Remark 2.3. From the proof we get a slightly stronger statement, namely that

$$\sum_{S\in E}c(S) \leq H_{\max_{S\in\{S_1,\ldots,S_m\}}|S|}\sum_{S\in E^*}c(S).$$

We also observe that the runtime is polynomial, namely it is

$$O\left(m^2+\sum|S_i|\right)$$

The Analysis is best possible: Take a set of n elements and take a singleton set for each element. Assign the weight $\frac{1}{n-(i-1)}$ for the singleton set i (assign the weight $1+\varepsilon$ to the whole set). Then the approximation ratio is $\frac{H_n}{1+\varepsilon}$. Even in the unweighted case, there is an example with 12 sets that shows that the analysis was best possible.

Question: What happens if $c:\{S_1,\ldots,S_m\}\to\mathbb{R}$, i.e. negative weights for some sets are possible?

An idea would be to take all sets with negative weight and run the greedy algorithm on the remaining algorithms. This has approximation ratio ∞.

Remark 2.4. An interesting case for the problem is if $|S_i|\leq 2$. Then this is an edge cover problem (in P). Already for a bound of 3, the problem becomes NP-hard.

2.2 Vertex Cover: Special case of set Cover

Theorem 2.5
There exists a 2*-approximation for* Minimum Vertex Cover*.*

Proof. Compute a maximal matching and take all endpoints of the matching into a set S. Then S is a vertex cover of size $\leq 2OPT$.

Observation: For each matching edge at least one endpoint must be in a vertex cover. ∎

Unless P=NP 1.36 is a lower bound on the approximation ratio. One doesn't know yet whether a better constant exists.

2.3 k-Center

Input: A finite set X, a metric $dist : X \times X \to \mathbb{R}, K \in \mathbb{N}$

Output: A set $C \subseteq X$ with $|C| = K$ s.t. $\max_{x \in X} \min_{c \in C} dist(x, c)$ is minimum

k-Center can be solved in $O(k|X|^{k+1})$, i.e. we have polynomial runtime if $k = const$.

Consider X to be the vertices of a complete graph. Then We can identify the metric with $dist : E(G) \to \mathbb{R}_+$.

Definition 2.6. Given a graph $G = (V, E)$ one says that $D \subseteq V(G)$ is a **dominating set** if for all $v \in V(G) \setminus D$ there is some $d \in D$ with $\{v, d\} \in E(G)$. The **domination number** of the graph is defined as

$$dom(G) \coloneqq \min\{|D| : D \text{ is a dominating set in} G\}.$$