Technische Universität München (I7)
Prof. J. Křetínský / Dr. C.H. Broadbent

Summer term 2016
18. Juli 2016

Solution

Computational Complexity – Homework 11

Discussed on TBA: (Please see Website News section for information as an email needs to be sent to me to arrange next week's tutorial time(s)).

Exercise 11.1

(a) Modify the approximation algorithm for vertex cover shown in the lecture (lecture 18, slide 19) such that it always computes an optimal solution if the given graph is a disjoint union of linear chains.

(b) Consider the following algorithm for approximating an optimal vertex cover:

While G has edges choose any node v of maximal degree of G; add it to C; and remove v and all edges connected to it from G.

- Quantify the approximation this algorithm obtains on the following graph:

  $V := \{a_1, a_2, a_3, a_4\} \cup \{b_1, b_2\} \qquad E := \{\{a_i, a_i\} | 1 \leq i \leq 4\} \cup \{\{a_i, b_j\} | 1 \leq i \leq 4, 1 \leq j \leq 2\}$

- Can you generalize the graph from above to show that the approximation error can be as large as $\approx \ln |V|$?

Exercise 11.2

Show that, if Sat $\in$ $\mathbf{PCP}(r(n), 1)$ for some $r(n) = o(\log n)$, then $\mathbf{P} = \mathbf{NP}$.

Solution: Assume that Sat $\in$ PCP$(r(n), 1)$. Then there is some poly-time Turing machine V (time bound $T(n)$) s.t. for every 3CNF ϕ (obviously, it is enough to consider only these formulae)

- if ϕ is satisfiable, then

$$\exists \pi \, : \, \#\{\rho \in \{0,1\}^{r(|\phi|)} \mid V(x, \rho, \pi) = 1\} = 2^{r(|\phi|)}$$

- if ϕ is not satisfiable, then

$$\forall \pi \, : \, \#\{\rho \in \{0,1\}^{r(|\phi|)} \mid V(x, \rho, \pi) = 1\} \leq 1/2 \cdot 2^{r(|\phi|)}$$

where V access at most $q \in \mathcal{O}(1)$ bits from the possible proof π. In particular, the queries done by V on a given input ϕ are assumed to be *nonadaptive* by definition, i.e., the locations $i_1(\phi,\rho),\dots,i_q(\phi,\rho)$ at which V queries π are completely determined by ϕ and ρ. For every ϕ and $\rho \in \{0,1\}^{r(|\phi|)}$ we may consider the Boolean function $f_{\phi,\rho}(x_{i_1(\phi,\rho)}, x_{i_2(\phi,\rho)},\dots,x_{i_q(\phi,\rho)})$ which is true iff V accepts x on random bits ρ and query-answers $x_{i_1(\phi,\rho)}, x_{i_2(\phi,\rho)},\dots,x_{i_q(\phi,\rho)}$.

As q is constant and every $f_{\phi,\rho}$ can be evaluated in polynomial time (simply run V), we can calculate in time $\mathcal{O}(T(n))$ also the truth table of $f_{\phi,\rho}$. From this truth table (which is of constant size $\{0,1\}^{2^q}$), we can compute a 3CNF formula $\psi_{\phi,\rho}$ which represents $f_{\phi,\rho}$ in time exponential in q, i.e., constant time w.r.t. the input.

Define now

$$\psi := \bigwedge_{\rho\in\{0,1\}^{r(|\phi|)}} \psi_{\phi,\rho}.$$

Then ψ is satisfiable iff ϕ is satisfiable. Computing ψ amounts to simulating V at most 2^q times for every $\rho \in \{0,1\}^{r(|\phi|)}$ which takes time $\mathcal{O}(2^{r(|\phi|)} \cdot T(|\phi|))$.

ψ uses at most $\mathcal{O}(2^{r(|\phi|)})$ distinct variables. So, we need at most $\mathcal{O}(r(|\phi|))$ bits for representing a single variable, hence,

$$|\psi| \in \mathcal{O}(2^{r(n)} \cdot r(|\phi|)) \subseteq \mathcal{O}(2^{2r(|\phi|)}),$$

i.e., there is some constant $c > 1$ s.t. $|\psi| \le c \cdot 2^{2r(|\phi|)}$ if $|\phi| \ge n_0$ for some n_0.

Now, by making n_0 even larger, we may assume that $r(n) \le 1/4 \log n$ for all $n \ge n_0$, i.e.,

$$|\psi| \le c \cdot 2^{1/2 \cdot \log|\phi|} = c \cdot |\phi|^{1/2}.$$

So, every formula of length at least n_0 is equivalent to some formula of length $cn^{1/2}$ w.r.t. satisfiability.

Assume now, we apply this construction at most $\log n$-times (i.e., we immediately stop if we obtain a formula of length n_0) where n is the length of the original formula. Every reduction takes time polynomial in the original formula, so the total time is also bounded by some polynomial. Consider the length of the resulting formula assuming that it still is of length at least n_0:

$$n \to cn^{1/2} \to c(c^{1/2}n^{1/4}) \to c(c^{1/2}c^{1/4}n^{1/8}) \to \dots \to c^{\sum_{i=0}^{-1+\log n} 2^{-i}} n^{2^{-\log n}} \le c^2 n^{1/n}.$$

Now, as $n^{1/n} = e^{1/n \cdot \log n}$ goes monotonically to 1 for $n \to \infty$, we can choose n_0 even so large, that $c^2 \cdot n_0^{-n_0} \le n_0$. Within polynomial time we therefore can reduce the original formula ϕ to a formula of length at most n_0. Obviously, we can decide for every formula of length at most n_0 in constant time whether it is satisfiable or not.

Exercise 11.3

Prove that QuadEq is **NP**-complete.

Solution: We show 3Sat $\le_p$ QuadEq.

Assume $\phi = C_1 \wedge \dots \wedge C_m$ is a 3CNF-formula over the variables $x_1,\dots,x_n$.

Note that for every clause $C = l_1 \vee l_2 \vee l_3$ we have

$$l_1 \vee l_2 \vee l_3 \text{ iff } \neg(\bar{l}_1 \wedge \bar{l}_2 \wedge \bar{l}_3).$$

Note that $\wedge$ and multiplication on the field $\mathbb{F}_2$ coincide. We therefore add for every clause the equation:

$$\bar{l}_1 \cdot \bar{l}_2 \cdot \bar{l}_3 = 0.$$

In order to reduce the degree of every monomial to two, we replace $\bar{l}_2 \cdot \bar{l}_3$ by the auxiliary variable $[\bar{l}_2\bar{l}_3]$ and add the equation

$$[\bar{l}_2\bar{l}_3] = \bar{l}_2 \cdot \bar{l}_3.$$

Finally, we add the usual equations which ensure that associated literals are consistent, i.e.,

$$x + \overline{x} = 1.$$

Example: for $\phi = (x \vee y \vee z) \wedge (\overline{x} \vee y \vee \overline{z})$ we obtain the two equations:

$$\begin{aligned} 0 &= \overline{x} \cdot \overline{y} \cdot \overline{z} \\ 0 &= x \cdot \overline{y} \cdot z \end{aligned}$$

Introducing the auxiliary variables for quadratic terms, we obtain:

$$\begin{aligned} 0 &= \overline{x} \cdot [\overline{y}\overline{z}] \\ 0 &= x \cdot [\overline{y}z] \end{aligned}$$

$$\begin{aligned} 0 &= \overline{y} \cdot \overline{z} + [\overline{y}\overline{z}] \\ 0 &= \overline{y} \cdot z + [\overline{y}z] \end{aligned}$$

To these equations, we add the constraints for the literals:

$$\begin{aligned} 1 &= x + \overline{x} \\ 1 &= y + \overline{y} \\ 1 &= z + \overline{z} \end{aligned}$$

Consider the satisfying assignment $x = 0, y = 1, z = 0$. We then have

$$\begin{aligned} \overline{x} &= 1 + x = 1 + 0 = 1 \\ \overline{y} &= 1 + y = 1 + 1 = 0 \\ \overline{z} &= 1 + z = 1 + 0 = 1 \\ [\overline{y}\overline{z}] &= \overline{y} \cdot \overline{z} = 0 \cdot 1 = 0 \\ [\overline{y}z] &= \overline{y} \cdot z = 0 \cdot 0 = 0 \\ \overline{x} \cdot [\overline{y}\overline{z}] &= 1 \cdot 0 = 0 \\ x \cdot [\overline{y}z] &= 0 \cdot 0 = 0 \end{aligned}$$

i.e., the satisfying assignment yields also a solution of the quadratic equation system.

Similarly, one checks that any solution of the quadratic systems yields a satisfying assigment for the original formula.

Exercise 11.4

Consider the following problem:

Input:	A matrix $A \in \mathbb{Q}^{m \times n}$, a vector $b \in \mathbb{Q}^m$.
Target:	Determine the maximal number of equations in $Ax = b$ which can simultaneously be satisfied by some $x \in \mathbb{Q}^n$.

Show that there is a constant $\rho < 1$ such that approximating the maximal size is **NP**-hard.

Solution: Consider any 3CNF-formula $\phi = C_1 \wedge \ldots \wedge C_m$ over the variables $x_1, \ldots, x_n$. We may assume that every clause consists of exactly three literals by simply repeating some literal if necessary.

Now, transform a clause C_i into three equations stating that exactly one, resp. two, resp. three literals are true. For instance, if $C = x \vee \overline{y} \vee x$, then add the equations

$$\begin{aligned} 1 &= x + 1 - y + x \\ 2 &= x + 1 - y + x \\ 3 &= x + 1 - y + x \end{aligned}$$

We do this for every clause of ϕ, so some equation might be appear several times in the final equation $b = Ax$ system.

Consider now an assignment $\alpha \in \{0,1\}^n$ s.t. exactly k clause of ϕ are satisfied. Then exactly k equations of $Ax = b$ are satisfied for $x := \alpha$.

But how can we rule out that there some $x \in \mathbb{Q}^n$ s.t. more than k equations in $Ax = b$ are satisfied? We simply add "enough incentive" to only consider Boolean vectors x: Any $x \in \mathbb{Q}^n$ can satisfy at most m equations by construction of $Ax = b$. So, for every variable x_i add <u>$m+1$ copies</u> of the equations

$$\begin{aligned} 0 &= x_i \\ 1 &= x_i \end{aligned}$$

We write $A'x = b'$ for the resulting system.

Now, any $x \in \{0,1\}^n$ will always satisfy at least the $n \cdot (m+1)$ equations of $A'x = b'$ associated with variables. But every $x \notin \{0,1\}^n$ can satisfy at most $m + (n-1)\cdot(m+1) = n\cdot(m+1) - 1$ equations: at most m equations associated with the m clauses of ϕ; and as at least one component of x is neither 0 nor 1, at most $(n-1)(m+1)$ equations associated with variables.

For the following, let $k_{\max}$ be the number of maximal clauses simultaneously satisfiable in ϕ. Similarly, let $n_{\max}$ be the maximal number of equations simultaneously satisfiable in $A'x = b'$. Then $n_{\max} = k_{\max} + n(m+1)$.

Now, we know that there is some $\rho < 1$ s.t. it is **NP**-hard to decide whether $k_{\max} < \rho \cdot m$ or $k_{\max} = m$ for arbitrary ϕ. Via our reduction, we can construct for every ϕ the equation system $A'x = b'$ s.t. this becomes the question whether $n_{\max} < \rho m + n(m+1)$ or $n_{\max} = m + n(m+1)$. Hence, if we could approximate $n_{\max}$ (in polynomial time) for any $\rho' < 1$, we could choose ρ' so close to 1 that

$$\rho m + n(m+1) \leq \rho'(m + n(m+1)).$$

Let $n_{\approx}$ now be the ρ'-approximation of $n_{\max}$. If $n_{\approx} \geq \rho'(m + n(m+1)$, then by choice of ρ' we also know that $n_{\approx} \geq \rho m + n(m+1)$ and, thus, can conclude that $k_{\max} \not< \rho m$, i.e., $k_{\max} = 1$.

Remark: For a reduction from vertex cover/independent set see "hardness of approximate optima in lattices, codes, and systems of linear equations" by Arora, Babai and Stern.

<u>Exercise 11.5</u>

We consider the optimization variant of the KNAPSACKPROBLEM:

Input:	Values $v_1, \dots, v_n$, weights $w_1, \dots, w_n$ and a weight bound W, all natural numbers representable by n bits.
Target:	Compute the maximal total value attainable by any selection S of total weight at most W, i.e.,

$$v_{\text{opt}} := \max\{\sum_{i \in S} v_i \mid S \subseteq \{1, 2, \dots, n\} \wedge \sum_{i \in S} w_i \leq W\}.$$

(a) In Exercise 3.2(c) we have discussed a pseudo-polynomial algorithm which solves this problem in time $\mathcal{O}(nW)$. Similarly, design an algorithm which finds the maximal total value by computing an array A with

$$A[j, v] = \min\{W + 1, \sum_{i \in S} w_i \mid S \subseteq \{1, 2, \dots, j\} \wedge \sum_{i \in S} v_i = v\}.$$

Your algorithm should be polynomial in n and $V := \sum_{i=1}^{n} v_i$.

(a') Modify your algorithms so that it runs in time polynomial in n and v_{opt}.

(b) Assume you replace all values v_i by $v'_i := \lfloor v_i / 2^k \rfloor$ for some fixed $k \geq 0$, i.e., you remove the k least significant bits. The weights w_i and the weight limit W stay unchanged. Let v_{opt}, resp. v'_{opt} be the optimal value for the original resp. reduced instance.

We take $v'_{\text{opt}} \cdot 2^k$ as an approximation for v_{opt}.

- Show that $v_{\text{opt}} \geq v'_{\text{opt}} 2^k$. What is the approximation error in the worst case?
- Choose k s.t. the approximation error is at most $\epsilon > 0$. Show that for this k the algorithm runs in time polynomial in n and $1/\epsilon$.

Solution: (See also "Combinatorial Optimization" by Papadimitriou and Steiglitz, section 17.3 "approximation schemes")

(a) We may assume that $v_i > 0$ as all items of zero value can be simply discarded. We may initialize the array by $A[0..n, 0] := 0$ and $A[1..n, 0..V] := W+1$. Then for $j = 0, 1, 2, \dots, n-1$:

- for $v = 1, 2, \dots, V$: (test for every possible value what is the minimal weight of a selection $S \subseteq [j]$)
  - if $A[j, v] \leq W$: (only if we are still under the weight limit, try to add item $j+1$)
    - $A[j+1, v] := \min\{A[j+1, v], A[j, v]\}$ (we do not add item $j+1$ to the selection associated with $A[j, v]$)
    - $A[j+1, v+v_{j+1}] := \min\{A[j+1, v+v_{j+1}], A[j, v] + w_{j+1}\}$ (we add item $j+1$ to the selection)

Finally, scan $A[n, 1..V]$ for the optimal value v_{opt}, i.e., the largest v s.t. $A[n, v] \leq W$.

Note that we have to add and compare in the worst case numbers of length $2n$ bits as we assumed that all values and weights are n-bit numbers. So, the algorithms runs in time $\mathcal{O}(n^2 V)$.

(b) We replace the two-dimensional array A by an array B of binary trees $B[j]$: If $A[j,v] < W+1$, we insert the path corresponding to the binary representation of v into $B[j]$ (if it doesn't exist already) and store in the last node of this path $A[j,v]$. Obviously, every tree $B[j]$ has at most v_{opt} many leaves and height at most $\log V \leq 2n$. Hence, every tree $B[j]$ has at most $2n \cdot v_{\text{opt}}$ leaves where inserting and updating entries takes at most $2n$ time. This leads to a running time of $\mathcal{O}(n^3 v_{\text{opt}})$.

(c) Assume that S resp. S' is an optimal selection attaining v_{opt} resp. v'_{opt}. These selections have to exist.

As S is optimal, we have

$$v_{\text{opt}} = \sum_{i\in S} v_i \geq \sum_{i \in S'} v_i = \sum_{i\in S'} v'_i \cdot 2^k + (v_i - v'_i \cdot 2^k) = 2^k \cdot \left(v'_{\text{opt}} + \sum_{i\in S'} \underbrace{v_i/2^k - \lfloor v_i/2^k \rfloor}_{\in[0,1)} \right) \geq v'_{\text{opt}} \cdot 2^k$$

In particular:

$$\frac{v_{\text{opt}} - v'_{\text{opt}}}{v_{\text{opt}}} = \frac{2^k \sum_{i\in S'} v_i/2^k - \lfloor v_i/2^k \rfloor}{v_{\text{opt}}} \leq \frac{2^k \cdot n}{v_{\text{opt}}} =: \epsilon, \textit{i.e.}, \frac{v_{\text{opt}}}{2^k} = \frac{n}{\epsilon}.$$

So, the running time can also be expressed w.r.t. n and $1/\epsilon$:

$$\mathcal{O}(n^3 \cdot v'_{\text{opt}}) \subseteq \mathcal{O}(n^3 \cdot v_{\text{opt}}/2^k) \subseteq \mathcal{O}(n^4 \cdot 1/\epsilon).$$