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

Summer term 2016
19.4.2016

Solution

Computational Complexity – Homework 2

Discussed on Monday, 25.4.2016.

Exercise 2.1

Let DNF-SAT be the set of all satisfiable boolean formulae in disjunctive normal form.

- Show that DNF-SAT is in **P**.

Let 2SAT be the set of all satisfiable boolean formulae in conjective normal form where every clause consists of at most two literals.

- Show that 2SAT is in **P**.

  Remark: In fact, 2SAT can be decided in $\mathbf{SPACE}((\log n)^2)$, resp. in **NL**.

Solution: A formula in DNF is satisfiable iff some conjunctive clause does not contain contradictory literals. This can easily be checked in polynomial time by checking each conjunctive clause in turn.

Now consider a 2SAT formula of the form

$$\phi := C_1 \vee C_2 \vee \cdots \vee C_k$$

where each of the C_i contains precisely two literals. (Note that even a constant-space machine (i.e. a finite automaton) can parse such formulae and so we can certainly validate the form of the input in logspace or polynomial time.)

The formula ϕ induces an *implication graph* G_ϕ that has literals together with the symbol $\bot$ as nodes and directed edges of the form:

$$l_1 \longrightarrow l_2$$

whenever there is a clause in ϕ of the form $\overline{l_1} \vee l_2$ (i.e. whenever there is a clause equivalent to $l_1 \to l_2$).

Now consider the following two rules that define new edges in the graph based on previously existing edges.

$$\frac{l_1 \longrightarrow l_2 \qquad l_2 \longrightarrow l_3}{l_1 \longrightarrow l_3} \qquad\qquad \frac{l_1 \longrightarrow l_2 \qquad l_1 \longrightarrow \overline{l_2}}{l_1 \longrightarrow \bot}$$

So in particular repeated applications of the first rule computes the transitive closure of the graph. If one interprets $\bot$ as 'false' or 'contradiction', then it is clear that these rules preserve the interpretation of edges as implications: that is if there is an edge $l_1 \longrightarrow l_2$ (where l_2 may be $\bot$ as well as a literal), then $\phi \vDash l_1 \to l_2$.

Let us call the graph that results from adding all possible edges to G_ϕ via the rules the graph $\hat{G}_\phi$.

We claim that ϕ is satisfiable iff it is the case that for every variable x, either x or $\overline{x}$ has *no* edge to $\bot$ in $\hat{G}_\phi$.

First suppose that for some variable x both $x \longrightarrow \bot$ and $\overline{x} \longrightarrow \bot$. Then by the preservation of the interpretation of edges $\longrightarrow$ it must be the case that setting x to true implies a contradiction and setting x to false also does, and thus ϕ must be unsatisfiable.

Now suppose that for every variable x_i there is a literal $l_i \in \{x_i, \overline{x_i}i\}$ such that l_i has no edge to $\bot$. We need to show that ϕ has a satisfying assignment.

In order to do this we construct a set S of literals that when set to true constitute a satisfying assignment.

We construct this set inductively. We begin with $S_0 := \{l_1\}$ (although we could start with any l_i).

We then define

- $S_{j+1} := \{l | l' \longrightarrow l \text{ for some } l \in S_j \text{ in } \hat{G}_\phi\}$ if this set contains an l not already in S_{j+1},
- otherwise $S_{j+1} := S_j \cup \{l_i\}$ for some l_i not already in S_{j+1},
- otherwise we are done and set $S := S_j$.

Due to the second item, S contains either x_i or $\overline{x_i}$ for all variables x_ii. Note further that since $\hat{G}_\phi$ is transitive-closed and since by assumption none of the l_i have edges to $\bot$, it follows that no literal in S has an edge to $\bot$ and also that at most one of x_i and $\overline{x_i}$ belongs to S for each variable x_i. Thus S is indeed valuation.

S must, moreover, be a satisfying valuation. For suppose not. Then there must be some clause $C_j = \{l, l'\}$ such that $\overline{l}, \overline{l'} \in S$. But then there is an edge in $\hat{G}_\phi$ from l to $\overline{l'}$ (and also from l' to $\overline{l}$). Thus we would have $\overline{l'} \in S$ (and also $\overline{l} \in S$) contradicting the fact that it does indeed define a valuation.

We can now show that 2SAT is in **P**. It suffices to construct $\hat{G}_\phi$ in polynomial time, since once we have constructed $\hat{G}_\phi$ (which is clearly polynomially big in the size of ϕ), it can be checked in polynomial time whether or not there exists a variable x_i such that both x_i and $\overline{x_i}$ have edges to $\bot$.

The machine maintains a worklist W of edges. Each edge in this worklist must be 'fully processed' before being added to the graph being constructed. This ensures that each edge is only processed once. Let E denote the set of edges added so-far to the graph being constructed.

The machine proceeds as follows:

- Parse ϕ and add all edges belonging to G_ϕ to W. The set E begins as the empty set.

- Pick an edge e in W. By comparing it to each edge e' in E, apply the two edge-construction rules. If a new edge e'' is so produced (that does not already belong to E or W), then add it to W. The edge e is then removed from W and added to E.
- Repeat the previous step until there are no more edges to add.

If there are n literals, then at most $O(n^2)$ edges can be produced. Each edge-construction rule is applied at most once for each pair of edges (since one edge must belong to the work-list and is removed from the work-list as soon as this is completed) and so at most $O(n^4)$ applications of the rules are made. When a rule is applied, any generated edge must be compared to all of the other edges previously produced to avoid re-adding an edge to W. This brings the complexity up to $O(n^5)$ in the number of literals.

Since the number of literals is bounded by the size of the formula, such a machine must indeed run in polynomial time, as required.

To see that the problem is in **NL**, we avoid explicitly constructing $\hat{G}_\phi$ and instead consider paths in G_ϕ. (G_ϕ is effectively represented by the formula ϕ on the machines input tape). The formula in *unsatisfiable* if there exists a variable x such that there is a path from x to a literal l and a literal $\bar{l}$ and also a path from $\overline{x}$ to a literal l' and a literal $\overline{l'}$. A non-deterministic machine can guess the variable x and the literals l and l' and then non-deterministically check reachability in G_ϕ (reachability in **NL** has been seen in the lectures. This shows that 2UNSAT is in **NL**. Since **NL** is closed under complement, 2SAT must also belong to this class.

Exercise 2.2

A *clique* in a graph is a set of vertices that are all connected to each other with the graph edges. Let CLIQUE = $\{\langle G, k\rangle \mid$ graph G has a clique of k vertices$\}$. Show the following:

(a) INDSET $\leq_p$ CLIQUE

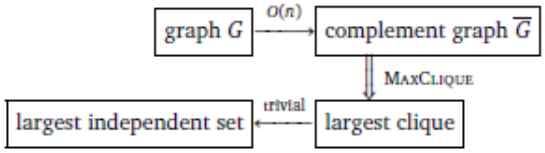
(b) CLIQUE $\leq_p$ INDSET

(c) $3 - \text{SAT} \leq_p$ CLIQUE

(d) CLIQUE is **NP**-complete.

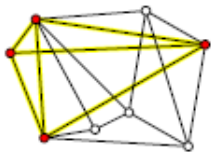
Solution:

There is an easy proof that MaxClique is NP-hard, using a reduction from MaxIndSet. Any graph G has an *edge-complement* $\overline{G}$ with the same vertices, but with exactly the opposite set of edges—(u, v) is an edge in $\overline{G}$ if and only if it is *not* an edge in G. A set of vertices is independent in G if and only if the same vertices define a clique in $\overline{G}$. Thus, we can compute the largest independent in a graph simply by computing the largest clique in the complement of the graph.



16.6 Maximum Clique Size (from 3SAT)

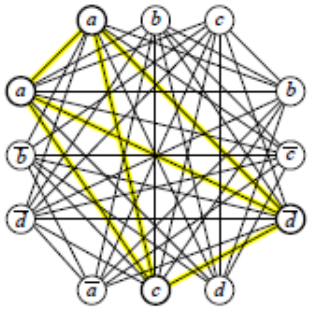
The last problem I'll consider in this lecture is a graph problem. A *clique* is another name for a complete graph. The *maximum clique size* problem, or simply MAXCLIQUE, is to compute, given a graph, the number of nodes in its largest complete subgraph.



A graph with maximum clique size 4.

I'll prove that MAXCLIQUE is NP-hard (but not NP-complete, since it isn't a yes/no problem) using a reduction from 3SAT. I'll describe a reduction algorithm that transforms a 3CNF formula into a graph that has a clique of a certain size if and only if the formula is satisfiable.

The graph has one node for each instance of each literal in the formula. Two nodes are connected by an edge if (1) they correspond to literals in different clauses and (2) those literals do not contradict each other. In particular, all the nodes that come from the same literal (in different clauses) are joined by edges. For example, the formula $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$ is transformed into the following graph. (Look for the edges that *aren't* in the graph.)



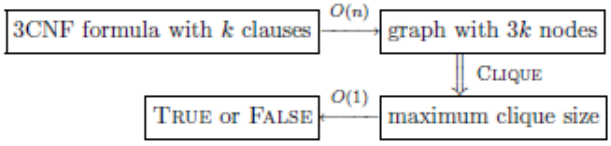
A graph derived from a 3CNF formula, and a clique of size 4.

Now suppose the original formula had k clauses. Then I claim that the formula is satisfiable if and only if the graph has a clique of size k.

1. ***k*-clique $\implies$ satisfying assignment:** If the graph has a clique of k vertices, then each vertex must come from a different clause. To get the satisfying assignment, we declare that each literal in the clique is true. Since we only connect non-contradictory literals with edges, this declaration assigns a consistent value to several of the variables. There may be variables that have no literal in the clique; we can set these to any value we like.

2. **satisfying assignment $\implies$ *k*-clique:** If we have a satisfying assignment, then we can choose one literal in each clause that is true. Those literals form a clique in the graph.

Thus, the reduction is correct. Since the reduction from 3CNF formula to graph can be done in polynomial time, so MaxClique is NP-hard. Here's a diagram of the reduction:



$$T_{3\text{SAT}}(n) \leq O(n) + T_{\text{MaxClique}}(O(n)) \quad \implies \quad T_{\text{MaxClique}}(n) \geq T_{3\text{SAT}}(\Omega(n)) - O(n)$$

Exercise 2.3

Argue that the following theorem on the linear speedup of Turing machine holds:

> Let $L \subseteq \{0,1\}^*$ be a language decided by a Turing machine M in time $T(n)$. Then, for any $c > 0$ there is Turing machine M' which decides L in time $T'(n) := cT(n)+n+C$ (with C some constant independent of L or c, e.g., $C \leq 10$ should work).

Remark: Fix any constant $m \in \mathbb{N}$. Then M' first compresses the input from size n to size $\lceil \frac{n}{m} \rceil$ on some auxiliary work tape. Then M' simulates m steps of M within at most 10 steps. (In fact, 6 steps should be sufficient.) Finally, choose the constant m in such a way that M' simulates M in time $T'(n)$.

Solution: M' behaves as follows:

(a) Compression phase:

M' compresses the input x using a vector alphabet, e.g., if M uses tape alphabet Γ, then Γ^m is contained in the tape alphabet of M' For every word $w \in \Gamma^*$, let $\chi(w) \in \Gamma^m$ be its compressed representation, i.e., first append as few as possible $\square$ such that $w\square^k$ has lenght divisible by m. Then let $\chi(w)$ be the word we obtain from $w\square^k$ by reading it as word over Γ^m.

In the compression phase, M' simply reads the input from left to right (excluding $\triangleright$ which we do not compress), counts in its head up to m, and remembers at the same time the last m symbols read from the input. Every time the counter hits m, M' writes the last m input symbols $x_i x_{i+1} \dots x_{i+m-1}$ to a auxiliary tape using the symbol $(x_i, x_{i+1}, \dots, x_{i+m-1})$. At the end, the context of the auxiliary tape is $\chi(\triangleright x)$.

This step needs time $|x| + 2$ $(= |\triangleright x \square|)$.

(b) Simulation phase:

Then M' starts to simulate M on x. For this, the auxiliary tape becomes the new input tape. We forget the original input tape in the following.

M' first moves its (new) input head on the left-most position. This takes $\lceil \frac{n}{m} \rceil$.

All the time of the simulation the contents of the tapes of M and M' are in one-to-one correspondence: if $\triangleright w \in \Gamma^*$ is the content of the k-th tape of M, then $\chi(\triangleright w)$ is the content of the k-th tape of M'. Further, if the head of the k-th tape of M is on position i (counting from right), then the corresponding head of M' is on position $\lfloor \frac{i}{m} \rfloor$.

In order to obtain the linear speed up, M' simulates m steps of M within a single step. Note that within m steps every single head of M can move at most m positions to the right or left. Hence, it suffices for M' to remember for every tape the three symbols within one step of the corresponding head. M' can store this information within its control state. Note that for remembering these three symbols, every M' can move each of heads as follows: (i) one step to the right, (ii) two to the left (i.e., one step left of the original position of the head), (iii) finally one step to the right again (original position of the head). M' can store this finite information in its head. M' then updates its tape contents according to the transition relation of M within two steps (Why?!).

The total running time of M' is then at most $|x| + 2 + 6 \cdot \frac{T(|x|)}{m}$. So, taking m large enough, we obtain the required speedup.

Exercise 2.4

(a) Show that **EXPTIME**-hard problems exist. Use the idea of the universal Turing machine.

(b) Show that **EXPTIME**-complete problems exist. Use a modification of the previous result.

(c) Show that the same holds for **DTIME**(f), **NTIME**(f), **DSPACE**(f), **NSPACE**(f) for any constructible f.

Exercise 2.5

Let M be a Turing machine which computes a function $f : \{0,1\}^* \to \{0,1\}^*$. As mentioned in the lecture, we are basically interested in two resources, time and space, needed by M for computing $f(x)$ from the input x. Measuring time is straight-forward, we simply count the number of steps M does on input x. In the case of space, one is usually not interested in the space required for storing the input or the output, but only in the space required for computing the output from the input. One therefore defines:

A function $f : \{0,1\}^* \to \{0,1\}^*$ is computable in space $S(n)$ if there is a Turing machine M_f such that

(i) M_f computes f.

(ii) M_f does not write any blanks ($\Box$).

(iii) M_f never moves the head of the output tape to the left.

(iv) For every input x of length $n = |x|$ the total number of non-blank symbols on all *work* tapes is bounded from above by $S(n)$ in every step of the computation.

Similar to the definition of **DTIME**, we write $f \in \mathbf{DSPACE}(S)$ if there is a Turing machine which computes f in space $S'(n)$ for some $S' \in \mathcal{O}(S)$. Finally, a language $L \subseteq \{0,1\}^*$ is decided in $\mathbf{SPACE}(S)$ if its characteristic function f_L is computable in $\mathbf{SPACE}(S)$ (with $f_L(x) := 1$ if $x \in L$, and $f_L(x) := 0$ if $x \notin L$).

(a) Show that the function inc : $\{0,1\}^* \to \{0,1\}^*$ which increases x by one (interpreting x as a natural number via the lsbf-encoding) is computable in constant space $\mathcal{O}(1)$.

(b) How much space is needed to decide the language of palindromes?

(c) Show or disprove that we may strengthen condition (iii) to "M_f never moves the head of the output tape to the left and never overwrites a non-blank symbol on the output tape".

(d) Argue that if a function f is computable in space $S(n)$, then it is also computable in space $cS(n) + C$ for any $c \in (0, \infty)$ (with C some constant independent of f or c, e.g., $C \leq 10$ should work).

*(e) For those who know two-way finite automata:

Argue that every Turing machine using bounded space is basically a finite automaton with output.

Solution:

(a) The TM only needs to remember the carry bit in order to determine the i-th bit of the ouput given the i-th bit of the input.

(b) The TM behaves as follows on input x with $n := |x|$:

Set $i := 1$. While $i \leq n$ do: Read symbol x_i from the input. Store it in the control. Move input head to the end of the input, then i positions back. Check that $x_i = x_{n+1-i}$. Set $i := i + 1$.

Note that for storing i, the TM only needs space $\log n$. Further, counting up to i can also be done in space $\log n$.

(c) The TM simply writes to the output tape on position i only before moving the output head to position $i + 1$. It can use its control to simulate any previous write to the output tape position i.

(d) We compress the work tapes on-the-fly similar to the construction used in ex 2.1, i.e., the new TM stores in its control state a block of m symbols plus the position of the head for every work tape of the original machine.

Note that the construction of ex 2.1 cannot be applied directly as we cannot store a linear compressed copy of the input on some work tape if $S(n)$ is sublinear.

(e) If a TM uses finite space, we can use the control state to store this finite information. We therefore obtain a TM with input and output tape, but without any further tapes. This is basically a finite automaton with output which can read the input several times. Such finite automata are called two-way finite automata (2FA). It is known that 2FA and FA recognize the same languages.