RWTH Aachen **WS 2010**
Lehrgebiet Theoretische Informatik Exam
Kneis—Langer—Rossmanith—Sikdar 23.3.2011

Exact Algorithms

Here we give fairly detailed solutions to the problems in the first exam.

Solution to Exercise 1

A simple recursive branching algorithm may proceed as follows: Given sets U_1, U_2, which are empty in the very first call of the algorithm, the algorithm picks a set $S_i = \{u_1, u_2, u_3\}$ with $S_i \cap U_1 \cup U_2 = \emptyset$, i.e., membership status of all three elements u_1, u_2, u_3 is still open. If such a S_i exists, then the algorithm recursively branches on the six possible ways to add u_1, u_2, u_3 into U_1, U_2 in a way that neither $\{u_1, u_2, u_3\} \subseteq U_1$ nor $\{u_1, u_2, u_3\} \subseteq U_2$. Otherwise, if no such S_i exists, then for each S_i there is at least one element $u_i \in S_i$ with $u_i \in U_1 \cup U_2$. The remaining problem can then be expressed as a 2-SAT formula and solved in polynomial time as follows: First iteratively apply the following reduction rules until application is no longer possible:

- Remove all S_i that are already split by U_1, U_2.
- If there is a S_i with $S_i \subseteq U_1$ or $S_i \subseteq S_2$, return *false*.
- If there is a S_i with $|S_i \setminus (U_1 \cup U_2)| = 1$ and S_i is not split yet, add the remaining single element to the correct U_1 or U_2 to make S_i be split.

Once these reduction rules have been applied, $|S_i \setminus (U_1 \cup U_2)| = 2$ for all S_i. We can now transform the problem into a 2-SAT instance, where U_1 contains the variables set to 0 and U_2 is the set of variables assigned 1. For each $u \in U$, we will create a variable x_u. For each $S_i = \{u_1, u_2, u_3\}$, where w.l.o.g. $u_1 \in (U_1 \cup U_2)$, we now create a clause C_i. If $u_1 \in U_1$, we let $C_i = \{x_{u_2}, x_{u_3}\}$. If $u_1 \in U_2$, we let $C_i = \{\neg x_{u_2}, \neg x_{u_3}\}$.

We now show that the algorithm is correct and yields the claimed running time. Since the algorithm can easily verify that a partition U_1, U_2 of U splits each S_i, the algorithm never says *true* on no-instances. If therefore suffices to prove that the algorithm says *true* on yes-instances. Let's say that (U_1', U_2') *extends* (U_1, U_2), if $U_i \subseteq U_i'$ for $i = 1, 2$, U_1, U_2. We prove the stronger claim that the algorithm, given sets U_1, U_2, finds a solution U_1', U_2' that extends U_1, U_2 if such a solution exists at all. Since each solution extends U_1, U_2 with $U_1 = U_2 = \emptyset$, the main claim then directly follows.

The proof is an induction over $|U \setminus (U_1 \cup U_2)|$. For the induction base, assume that U_1, U_2 are such that $|S_i \setminus (U_1 \cup U_2)| < 3$ for all S_i. It is not hard to see that the reduction rules are correct. For instance, if there is a S_i with $|S_i \setminus (U_1 \cup U_2)| = 1$ and, say $u_2, u_3 \in U_1$, then every solution U_1', U_2' extending U_1, U_2 has to have $x_1 \in U_2'$. Thus, we need to show that the reduction to 2-SAT is correct. For, let U_1', U_2' be a solution extending U_1, U_2 and for each $u \in U$, let $x_u := 0$ if $u \in U_1'$ and $x_u := 1$ if $u \in U_2'$. Now consider a set $S_i = \{u_1, u_2, u_3\}$ with one of the $u_j \in U_1$, say $u_1 \in U_1$. Since U_1', U_2' is a solution extending U_1, U_2, we know $u_1 \in U_1'$, and hence one of $u_2, u_3 \in U_2'$. Thus, the clause $C_i = \{x_{u_2}, x_{u_3}\}$ is satisfied by the assignment above. If $S_i = \{u_1, u_2, u_3\}$ is such that $u_j \in U_2$, say $u_1 \in U_2$, then at least one of u_2 or u_3 must be contained in U_1', since U_1', U_2' is a solution extending U_1, U_2. Therefore, the clause $C_i = \{\neg x_{u_2}, \neg x_{u_3}\}$ is satisfied. All in all, the assignment constructed above satisfies the formula. The converse direction follows analogously: If there is an assignment to the variables x_u, $u \in U$, that satisfies the constructed 2-CNF formula, then we let $U_1' = U_1 \cup \{\, u \mid x_u = 0 \,\}$ and $U_2' = U_2 \cup \{\, u \mid x_u = 1 \,\}$. Using the same arguments as above, we can show that each set S_i is correctly split by U_1', U_2'. Since 2-SAT can be solved in polynomial time, the running time is polynomial in $|U| = n$.

For the induction step, assume there is S_i with $|S_i \setminus (U_1 \cup U_2)| = 3$ and let U_1', U_2' be a solution extending U_1, U_2. Since U_1', U_2' splits S_i, neither $S_i \subseteq U_1'$ nor $S_i \subseteq U_2'$. Thus, there is a branch where

the algorithm calls itself with U_1'', U_2'' such that U_1', U_2' extends U_1'', U_2'', which in turn extends U_1, U_2. By the induction hypothesis, the algorithm will return *true* for the recursive call on U_1'', U_2'', and therefore the algorithm return *true* when called with U_1, U_2.

We now bound the size of the recursive search tree. The height of the search tree is easily seen to be bounded by $|U \setminus (U_1 \cup U_2)|$, since the algorithm stops branching once there is no set S_i with $|S_i \setminus (U_1 \cup U_2)| = 3$. Consider a call of the algorithm where there is a set S_i with $|S_i \setminus (U_1 \cup U_2)| = 3$. Then in each recursive branch, the algorithm adds the elements of S_i to $U_1 \cup U_2$, i.e., the size of $|U \setminus (U_1 \cup U_2)|$ decreases by 3 in each branch. Since there are six branches, this yields a branching vector of $(3, 3, 3, 3, 3, 3)$ measured in $|U \setminus (U_1 \cup U_2)| \leq n$ with a branching number of approx. $1.82 < 2$. Each single recursive call requires polynomial time. The algorithm therefore solves the problem in $O^*(1.82^n)$ steps.

Solution to Exercise 2

Given an instance (G, k) of Vertex Cover, construct an instance (G', k') of Dominating Set as follows. Define $V(G')$ as $V(G) \cup E(G) \cup \{x, y\}$, where $\{x, y\} \cap (V(G) \cup E(G)) = \emptyset$. For $u \in V(G)$ and $e \in E(G)$, there is an edge $\{u, e\}$ in the graph G' iff u is incident to e in G. Also x is connected to y, which is connected to all of $V(G)$. This completes the construction of G'. We claim that G has a vertex cover of size at most k iff G' has a dominating set of size at most $k' := k + 1$. For the forward direction, suppose that G has a vertex cover S of size at most k. Then note that $S \cup \{y\}$ is a dominating set in G' of size at most $k + 1$. Conversely, if G' has a dominating set S' of size at most $k + 1$, we may assume wlog that $\{x, y\} \cap S' = \{y\}$. Firstly note that one of x or y must be in the dominating set. If $y \notin S'$, then replace x by y to obtain a new solution of the same size. Since $y \in S'$, we may assume that every vertex of $V(G)$ in the solution dominates some vertex of $E(G)$. Since there are at most k vertices of $V(G)$ in S' and all of $E(G)$ is dominated, the graph G has a vertex cover of size at most k. Finally note that $|V(G')| = |V(G)| + |E(G)| + 2$ and $|E(G')| = 1 + |V(G)| + 2|E(G)|$. Hence if there is an algorithm deciding Dominating Set in time $O^*(2^{o(n+m)})$, there is one for Vertex Cover, contradicting ETH.

Solution to Exercise 3

Given a graph $G = (V, E)$, for $S \subseteq V$ define $\mathrm{OPT}[S]$ to be the chromatic number of the graph $G[S]$. Then $G[\emptyset] = 0$ and for $\emptyset \neq S \subseteq V$,

$$\mathrm{OPT}[S] = \min_X \{\mathrm{OPT}[S \setminus X]\} + 1,$$

where the minimum is taken over all maximal independent sets X of $G[S]$. All maximal independent sets in a graph with k vertices can be enumerated in time 2^k (by brute-force) and hence the time taken to compute the chromatic number is

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n,$$

modulo polynomial factors in the running time. If we use Moon and Moser's algorithm to compute the maximal independent sets in a graph (of which there are at most $3^{k/3}$), the exponential part of the running time of the DP algorithm is:

$$\sum_{k=0}^{n} \binom{n}{k} 3^{k/3} = (1 + 3^{1/3})^n < 3^n.$$

Solution to Exercise 4

We modify the problem slightly. Firstly, we number the boxes $1, \ldots, k$ and we allow each book to be placed multiple times in a box and across boxes. That is, a solution to our modified problem is k-tuple $(\sigma_1, \ldots, \sigma_k)$, where each σ_i is a sequence from $\{1, \ldots, n\}$ such that $\sum_{a \in \sigma_i} v(a) \leq B$ for all σ_i, and each $a \in \{1, \ldots, n\}$ is in some sequence σ_i. Note that any solution to the original problem is a solution

to the modified problem; and given any solution to the modified problem, one can remove multiple occurrences to obtain a solution to the original problem. This modification simply eases computations involving the terms of the IE-formula.

Define objects as stated in the hint. An object $(\sigma_1, \ldots, \sigma_k)$ satisfies property P_a for $a \in \{1, \ldots, n\}$ iff $a \in \sigma_i$ for some i. By Inclusion-Exclusion,

$$\left| \bigcap_{a \in [n]} P_a \right| = \sum_{X \subseteq [n]} (-1)^{|X|} \left| \bigcap_{a \in X} \bar{P}_a \right|.$$

For $X \subseteq [n]$, the term $|\bigcap_{a \in X} \bar{P}_a|$ counts the number of k-tuples $(\sigma_1, \ldots, \sigma_k)$ of sequences from $[n] \setminus X$ such that $\sum_{a \in \sigma_i} v(a) \leq B$ for all σ_i. Let $N[X]$ be the number of sequences σ from $[n] \setminus X$ such that $\sum_{a \in \sigma} v(a) \leq B$. Then

$$\left| \bigcap_{a \in X} \bar{P}_a \right| = N[X]^k.$$

It is important to note that the ordering of the boxes and the fact that books are allowed to appear multiple times makes the expression for the simplified problem so easy.

To compute $N[X]$, we use dynamic programming. Define $P_X(j)$ to be the number of sequences σ from $[n] \setminus X$ such that $\sum_{a \in \sigma} v(a) = j$. Then $P_X(0) = 1$ (the empty sequence avoids all of X and has sum 0). Also define $P_X(j) = 0$ for $j < 0$. For $j \geq 1$,

$$P_X(j) = \sum_{a \notin X} P_X(j - v(a)),$$

where $P_X(j - v(a))$ counts the number of such sequences with a as its last element. Now $N[X] = \sum_{i=0}^{B} P_X(i)$. The time taken to compute $P_X(j)$ for $0 \leq j \leq B$ is $O(nB)$ and the space required is $O(\max_a v(a))$. The total time required to compute all terms of the IE-formula is therefore $O^*(2^n \cdot nB)$.

Solution to Exercise 5

For each $\mathcal{F}' \subseteq \mathcal{F}$, we define a characteristic vector $v(\mathcal{F}') = (i_1, \ldots, i_n)$ with $i_j = |\{ F \in \mathcal{F}' \mid j \in F\}|$, i.e., the jth element i_j of $v(\mathcal{F}')$ specifies the number of sets in $\mathcal{F}'$ that contain element $j \in U$. A subset $\mathcal{F}' \subseteq \mathcal{F}$ is an exact set cover of U if $v(\mathcal{F}') = (1, \ldots, 1)$.

We can now compute a minimum exact cover in time $O^*(2^{m/2})$ as follows: First partition $\mathcal{F}'$ into two equally sized sets L and R (plus/minus one). Then compute the characteristic vector $v(L')$ for each $L \subseteq L'$ and the characteristic vector $v(R')$ for each $R \subseteq R'$. Since $|L| = |R| = O(m/2)$, this step takes time $O^*(2^{m/2})$. Now, for sets $L' \subseteq L$ and $R' \subseteq R$ we have that $L' \cup R'$ is an exact set cover for U iff $v(L') + v(R') = (1, \ldots, 1)$, where $v(L') + v(R')$ is the component-wise addition. To find such a pair L', R', we first sort the obtained vectors for L and R lexicographically. Sorting a set of size N takes $O(N \log N)$ comparisons, hence sorting the $O(2^{m/2})$ vectors of L and R takes time $O^*(2^{m/2})$ each. We now iterate over the vectors for L in increasing order. At the same time, we iterate over the vectors for R in decreasing order. At positions v_L and v_R, let i be the first position of $v_L + v_R = (v_1, \ldots, v_n)$ with $v_i \neq 1$. If no such position exists, an exact set cover has been found. If otherwise $v_i = 0$, we advance v_L to the next vector for L and proceed. If otherwise $v_i > 1$, we advance v_R to the previous vector for R and proceed.

If there is a solution, this technique will find a pair v_L, v_R with $v_L + v_R = (1, \ldots, 1)$: If i is the first position with of $(v_1, \ldots, v_n) = v_L + v_R$ with $v_i = 0$, then due to the lexicographical ordering $v'_L + v_R \neq (1, \ldots, 1)$ for all v'_L smaller than v_L and $v_L + v'_R \neq (1, \ldots, 1)$ for all v'_R larger than v_R. An analogous arguments holds for vectors over R that yield $v_i > 2$. Since in each step either one of the two vectors is changed, the running time for this "list" phase is $O^*(2^{m/2})$.