Institute of Theoretical Computer Science May, 2015
Emo Welzl, lecturer, and Jerri Nummenpalo, teaching assistant

Satisfiability of Boolean Formulas Spring 2015
Special Assignment Set 3 - Master Solution

Exercise 1 (Rigid k-colorings) *(50 points)*

a) Our algorithm will resemble the PPZ algorithm, but now in the graph setting. We will describe a Monte Carlo algorithm which has a certain success probability and repeatedly calling this algorithm until a solution is found will give us the desired running time bound in expectation. The input to our algorithm is a graph $G = (V, E)$ on n vertices for which there exists a rigid k-coloring and the output is a either a rigid k-coloring of the graph or the message FAIL .

We start with the description of our algorithm. First, we pick a random permutation $\pi \in \mathcal{S}_V$ of the vertices. Then, in the order described by the permutation, color each vertex v as follows. Consider the neighborhood of v. If all the k colors are present, the algorithm returns FAIL as the coloring can no longer be proper. If $k-1$ colors are present among the neighbors, then the color of v is forced to be the last available color in order to have a rigid k-coloring in the end. Therefore, we color v with the corresponding color. Otherwise we choose the color of v randomly from the available colors that don't conflict the so far attained coloring. If the process finishes without returning FAIL then we have constructed a proper coloring of the graph. We still have to check whether it is a rigid coloring and after the check we return the coloring or FAIL depending on the result.

It is clear that one run of the above procedure takes time polynomial in n. We want to bound the probability that our algorithm finds a rigid k-coloring. Towards that end, let C be a rigid k-coloring of the given graph that is promised to exist. In a similar fashion as how we encoded satisfying assignments for CNF formulas, we define an encoding of a coloring with respect to a permutation. For $\sigma \in \mathcal{S}_V$ we denote by $\text{enc}(C, \sigma)$ the encoding of the rigid k-coloring C (of the graph G) w.r.t. the permutation σ. More specifically, $\text{enc}(C, \sigma)$ will be a word over k symbols which is constructed by writing the colors of each vertex in the order specified by σ and by ignoring any vertices whose color is forced by the coloring which is attained so far.

Note that for a given σ the quantity $\text{enc}(C, \sigma)$ tells us how many random choices our algorithm has to correctly make in order to end up with the rigid k-coloring C in the end. The probability that the algorithm makes those choices correctly is at least $k^{-|\text{enc}(C,\sigma)|}$. For our analysis we also need a bound on the average encoding length over all possible permutations. For each vertex v of G let $M(v)$ be some fixed $k-1$ neighbors of v which all have different colors in C. Denote also by the brackets $[\cdot]$ the indicator function. The average encoding length is then

$$\begin{aligned}
\mathbb{E}_{\sigma \in \mathcal{S}_V}\left[|\text{enc}(C,\sigma)|\right] &= \frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} |\text{enc}(C,\sigma)| \\
&= \frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} n - \sum_{v \in V} [\text{color of vertex } v \text{ is forced under } \sigma] \\
&= n - \sum_{v \in V} \frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} [\text{color of vertex } v \text{ is forced under } \sigma] \\
&= n - \sum_{v \in V} \Pr(\text{color of vertex } v \text{ is forced under } \sigma) \\
&\leq n - \sum_{v \in V} \Pr(v \text{ is after every vertex in } M(v) \text{ in the permutation } \sigma) \\
&= n - \sum_{v \in V} \frac{1}{k} = n - \frac{n}{k}.
\end{aligned}$$

Now we can bound the success probability of our procedure:

$$\begin{aligned}\Pr(\text{algorithm returns } C) &= \sum_{\sigma\in\mathcal{S}_V} \Pr(\text{algorithm returns } C|\pi=\sigma)\cdot\Pr(\pi=\sigma)\\ &\geq \sum_{\sigma\in\mathcal{S}_V}\frac{1}{n!}\cdot k^{-|\mathrm{enc}(C,\sigma)|}\\ &\geq k^{-\frac{1}{n!}\sum_{\sigma\in\mathcal{S}_V}|\mathrm{enc}(C,\sigma)|}\\ &\geq k^{-(n-n/k)}\end{aligned}$$

where we did the same calculation as in Chapter 6, page 137 of the lecture notes slightly modified. We used Jensen's inequality to get the second to last line and for the last line we used the observation about the average encoding length of a coloring w.r.t. a random permutation.

As the success probability of our algorithm is at least $k^{-(n-n/k)}$ and every iteration of the algorithm takes time $\mathrm{poly}(n)$ we have in all a randomized algorithm with expected running time $\mathcal{O}(k^{n(1-1/k)}\mathrm{poly}(n))$.

b) We describe an algorithm similar to the algorithm *sb* from Chapter 7 of the lecture notes. Our algorithm is a depth first search in a search tree in which every node has at most $(k-1)$ many children. It works as follows. In each connected component of the graph select a vertex and fix some color for it. Then recursively do the following. Pick a vertex which has a neighbor whose color is already fixed and consider all the $k-1$ possibilities of coloring this new vertex. That is, in the search tree we move down by trying to fix the color of the new vertex one at a time to all the possible colors. If at any time some coloring would violate the condition for proper coloring we can ignore exploring the search tree further down that way. If at some point we arrive at a proper coloring we still need to test the coloring for being rigid. If it is we return this coloring and otherwise continue the search

As the search tree has depth n, every node has at most $k-1$ children and in every node the time spent is polynomial in n this gives us a deterministic $\mathcal{O}((k-1)^n\mathrm{poly}(n)$ algorithm.

c) Unfortunately this exercise was incorrectly posed, at least for the way it was meant to be solved. It relied on scanning the search ball around codewords, but this search ball is simply too big to be scanned in the time available.

We will still show that for $r=(1-1/k)n$ there exists a k-ary covering code of length n and covering radius r whose size is polynomial in n. By Chapter 7*.1, and parameters k,n,r' there exists a k-ary covering code with radius r' of length n and of size

$$\left\lceil \frac{n\ln(k)k^n}{\mathrm{vol}^{(k)}(n,r')}\right\rceil$$

where

$$\mathrm{vol}^{(k)}(n,r')=\sum_{i=0}^{r'}\binom{n}{i}(k-1)^i$$

is the size of the k-ary Hamming ball with radius r'. To have a code of size polynomial in n we want to find the values r' for which we have that $\mathrm{vol}^{(k)}(n,r')\geq k^n\cdot\mathrm{poly}(n)$. Towards that end note that the ratio of two consecutive terms in the sum expression of $\mathrm{vol}^{(k)}(n,r')$ is

$$\frac{\binom{n}{i+1}(k-1)^{i+1}}{\binom{n}{i}(k-1)^i}=\frac{n-i}{i+1}(k-1)$$

which is strictly bigger than one when

$$i<\frac{n(k-1)-1}{k}.$$

and (strictly) less than one when i is (strictly) bigger than that quantity. Therefore the sequence of terms in the sum of the volume is unimodal: They are increasing at first and then start decreasing

when reaching this threshold. If $r' \geq \lceil n \cdot \frac{k-1}{k} \rceil$, then the largest possible coefficient of the sum is present. Because of the binomial theorem we know that $\mathrm{vol}^{(k)}(n,n) = k^n$ so this largest coefficient has size at least $k^n/(n+1)$. Therefore by choosing $r = \lceil n \cdot \frac{k-1}{k} \rceil$ we have that

$$\mathrm{vol}^{(k)}(n,r) \geq k^n \cdot \mathrm{poly}(n).$$

We have established that having a radius bigger than $n \cdot \frac{k-1}{k}$ suffices to have a polynomial size k-ary covering code in the end. If we could scan the ball of radius r around one coloring in time $\mathcal{O}((1-k)^r \mathrm{poly}(n))$ then we would have the algorithm required. Unfortunately this does not work.

d) So far we have seen three algorithms on how to find a rigid k-coloring if it exists. The question arises if we could – instead of running our algorithms – decide whether a graph has a rigid k-coloring in some other, faster way. Björklund et al. [2009] demonstrate how to decide k-colorability in the general case in time $\mathcal{O}(2^n \mathrm{poly}(n))$ and our hope was that their methods could extend to the rigid case. We don't know how to do that, but we will briefly describe the main components of their proof.

The problems in the paper by Björklund et al. [2009] are formulated in terms of set theory. To rephrase the question of k-colorability in those terms, let $G = ([n], E)$ be a graph we want to inspect. Let $\mathcal{S}$ be the set of all independent sets of vertices in G. Note that $\mathcal{S}$ is a family of subsets of $[n]$. Now a k-coloring of G gives a covering the vertices of G with k independent sets. Conversely, given a cover of the vertices with k independent sets we can construct a proper k-coloring. So the question becomes, can we find $S_1, \ldots, S_k \in \mathcal{S}$ such that $S_1 \cup \ldots \cup S_k = [n]$? Such a collection of sets is called a *set cover* of size k of $\mathcal{S}$ or in other words a k-cover. Note that we allow repetition when choosing the S_i's – they could even all be same!

There are at most 2^n independent sets so we can construct $\mathcal{S}$ within the time bound we have at our disposal. The approach in Björklund et al. [2009] is then to compute the number of k-covers of $\mathcal{S}$ of size k. Denote this number by $c_k(\mathcal{S})$. If this number is strictly greater than zero then we know there exists a proper k-coloring. The main tool used to do this is the principle of inclusion-exclusion, which is an important concept in combinatorics. In our setting it works as follows. Let A_i denote the subsets of $\mathcal{S}$ which have k elements (independent sets), but none of which contains the element i. Let the complement, $\overline{A}_i$, be the set of subsets of $\mathcal{S}$ with k elements so that at least one of them contains the element i. We then have that

$$c_k(\mathcal{S}) = \left| \bigcap_{i=1}^{n} \overline{A}_i \right|$$

which by the inclusion-exclusion principle can be expressed as the sum

$$c_k(\mathcal{S}) = \sum_{X \in 2^{[n]}} (-1)^{|X|} \left| \bigcap_{i \in X} A_i \right|.$$

Above $\bigcap_{i \in \emptyset} A_i$ denotes the set of all k-subsets of $\mathcal{S}$. Thus, to compute the number of k-covers in the required time we need to be able to compute $\left|\bigcap_{i \in X} A_i\right|$ for all the 2^n choices of X in polynomial time. Björklund et al. [2009] give two ways to do this. In the more general case they show a way to compute all of these values in a total time of $\mathcal{O}(n2^n)$ by using the *fast zeta transform* or the *the fast Möbius transform* which achieves the desired running time. Specific to the coloring problem they also establish a recursive procedure to compute the values which allows one to compute the terms efficiently as well without resorting to other machinery.

Exercise 2

(i) We start by showing that the intersection of faces of the cube is again a face. We shall also call the empty set a face. Recall that a face $\phi_i \in \{0,1,*\}^n$ consists of all $\alpha \in \{0,1\}^n$ which match ϕ_i in the entries from $\{0,1\}$. Thus, the intersection of two faces $\phi_i, \phi_j \in \{0,1,*\}^n$ is empty if ϕ_i is 0 and ϕ_j is 1 on some coordinate and vice versa. In the case that there is no such conflict, the

intersection is specified by

$$(\phi_i \cap \phi_j)_k = \begin{cases} * & \text{if } (\phi_i)_k = (\phi_j)_k = * \\ 1 & \text{if } (\phi_i)_k = 1 \text{ and } (\phi_j)_k = * \\ 1 & \text{if } (\phi_i)_k = * \text{ and } (\phi_j)_k = 1 \\ 0 & \text{if } (\phi_i)_k = 0 \text{ and } (\phi_j)_k = * \\ 0 & \text{if } (\phi_i)_k = * \text{ and } (\phi_j)_k = 0 \end{cases}$$

for $k = 1, \ldots, n$. This is again a vector from $\{0, 1, *\}^n$ and therefore a face of the cube. The intersection of finitely many faces is just a repetition of this pairwise intersection and as taking then intersection is associative it is well defined.

We give a proof of the main claim where we translate the problem back to that of CNF formulas. Recall that each face corresponds to the unique clause which is unsatisfied by all the vertices in the face. The intersection of two faces therefore corresponds to the union of the corresponding clauses. If the intersection of two faces is nonempty then the union of the corresponding clauses don't have a complementary literal i.e. the union is not a trivially satisfiable clause. The intersection of all the faces corresponds to the union of all the clauses. If the pairwise unions are not trivially satisfiable, then so is not the union of all the clauses. This is because if the union contained a pair of complementary literals, then there would have to be two distinct clauses from which these complementary literals originated, but we assumed this not to be the case. This proves the claim.

(ii) Let $v \in \phi_1 \cap \phi_2 \cap \ldots \cap \phi_m$ be a vertex from $\{0, 1\}^n$ which is in the intersection of all the faces. Let $\overline{v}$ denote the antipodal vertex attained from v by flipping all the bits in v. Then we claim that $\overline{v}$ is a satisfying assignment for F. If not, then there existed $i \in \{1, \ldots, m\}$ such that $v \in \phi_i$ and $\overline{v} \in \phi_i$. But ϕ_i is a face so it has to be the whole cube. The only clause corresponding to the whole cube is the empty clause which we assumed no to be in F. Therefore this is a contradiction and F is indeed satisfiable.

(iii) Proof by induction on $|S|$. The case $|S| = 1$ is clear: just choose the whole cube. For general $|S|$ construct first a tiling for the first $|S| - 1$ elements. Then there exists a face which contains two elements. These two elements must differ in at least one position. We can split this face in two along this coordinate which gives us the desired tiling.

References

Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.