CSE 555 : Homework 4

Spring 2015

April 13, 2015

(1)

We use an intermediate problem, $\neq$ 4CNF SAT: Given a boolean formula with four literals per clause, is there a satisfying assignment in which every clause contains at least one true literal and at least one false literal?

We must first show that $\neq$ 4CNF SAT is NP-complete. A certificate is an assignment to the variables, for which it can be checked in polynomial time that every clause has both a true and a false literal. To show NP-hardness, we establish that 3CNF SAT $\leq_p \neq$ 4CNF SAT. Given any boolean formula $\phi$ in 3CNF, we add a single new variable $z$ and replace each clause $(\ell_1 \lor \ell_2 \lor \ell_3)$ of three literals by $(\ell_1 \lor \ell_2 \lor \ell_3 \lor z)$ to get a new formula $\psi$. If $\phi$ is satisfiable, then $\psi$ has a satisfiable assignment in which no clause has all literals true, because we set $z$ to false and use the assignment for $\phi$. Conversely if $\psi$ has a $\neq$-satisfying assignment, if $z$ is true then replace every truth assignment to a variable by its negation to get another $\neq$-satisfying assignment in which $z$ is false. Now the assignment to the variables of $\phi$ satisfies it. This reduction only adds one variable to each clause and so can be done in polynomial time. Hence $\neq$ 4CNF SAT is NP-complete.

Now we show that FLIP is NP-complete. Membership of FLIP in NP is immediate. A polynomial size certificate consists of a set of tile indices that correspond to the set of tiles to be flipped. Verification consists of flipping this set of tiles and then checking for every position $i, 1 \leq i \leq l$, if there is a tile with a zero in the $i$th position in the top string, and a tile with a zero in the $i$th position in the bottom string.

We reduce $\neq$ 4CNF SAT to FLIP to prove NP-hardness. Let $\phi$ be a Boolean formula in 4CNF with $n$ Boolean variables $x_1, \ldots, x_n$ and $l$ clauses. We construct $n$ tiles $[\frac{i}{n}], 1 \leq i \leq n$, corresponding to each Boolean variable. Top and bottom strings have $l$ symbols each. The $j$th position of $t_i$ has a 0 if $x_i$ appears in the $j$th clause. Similarly, the $j$th position of $b_i$ has a 0, if $\overline{x_i}$ appears in the $j$th clause of $\phi$. Otherwise, these positions have 1.

Suppose there is an assignment to $x_1, \ldots, x_n$ such that every clause of $\phi$ has both a true and a false literal. For each variable $x_i$ that is False, flip the corresponding tile, and leave all the other tiles as is. Each clause $C_j, 1 \leq j \leq l$, has a literal that is true and one that is false. So after flipping as described, there is a 0 in the top and the bottom in position $j$.

Conversely, if there is a FLIP solution, a variable is set false if its tile is flipped, true if not flipped. Then every clause contains both a true and a false literal, so this is a $\neq$-satisfying assignment.

Hence FLIP is NP-hard, and so we conclude that FLIP is NP-complete.

(Note: In any FLIP solution, you can flip all of the tiles and this must be another solution. This is why we used the intermediate problem of $\neq$ 4CNF SAT. However we could have done this whole question somewhat more simply by reducing 3CNF SAT directly to FLIP. To do this, proceed as above but add a tile that is all 1 on top and all 0 on the bottom. Actually this is equivalent to what we did.)

(2)

First we ask the oracle if $\langle T \rangle$ has a FLIP solution. If the answer is “yes”, then one by one, we change each 0 to a 1, and ask the oracle if the modified instance has a FLIP solution. If the the answer is “no” then we change
the 1 back to 0, otherwise we leave it. After this stage, for each position \( j, 1 \leq j \leq l \), there are exactly two tiles that have a 0 in the \( j \)th position, either in the top or the bottom string (indeed if there were more tiles with a 0 in the \( j \)th position, we could have turned them into 1, and still have a FLIP solution). If both the tiles have 0 in the \( j \)th position of the top string (or bottom string), then one of them must be flipped in the solution. But notice that it does not matter which one is flipped, since if we flip all the tiles in the solution, we still get a FLIP solution. Similarly, if one of the tiles has a 0 in the \( j \)th position in the top string, and another has a 0 in the \( j \)th position of the bottom string, then both the tiles must have same direction (both flipped or both unflipped) in the solution. So we need to figure out which tiles are “compatible” with each other. We call two tiles compatible if in a solution both of them are assigned same direction. Notice that compatibility is an equivalence relation. Again since there is a solution for the modified instance, we can compute this equivalence class in \( O(n^2) \) time by checking for every pair of tiles if they must have same direction or different direction. From this we can find a solution for the original instance.

We may need to query the oracle \( O(nl) \) time. Writing the question takes \( O(nl) \) times and reading the answer takes \( O(1) \) time. So the time complexity of the procedure is \( O(n^2l^2) \).

(A similar strategy works for problems like 3CNF SAT: Given a formula \( \phi \), we could choose a clause containing more than one literal, drop the literal from the clause and check if the simpler formula is still satisfied. If yes, proceed with that literal removed; otherwise replace the clause by one containing only that literal. At the end, you have all clauses containing only one literal, and can read out the assignment.)

(3)

**DUNDAS\(\in \text{NP}****: A polynomial size certificate consists of a subset \( E' \subseteq E \) of edges. The polynomial time verifier checks if it is a valid 2-factor, and the graph \((V, E')\) has at most \( c \) connected components (using breadth first search).

Next we show that \( \text{UHAMPATH} \leq_p \text{DUNDAS} \): Given a Graph \( G \) and two vertices \( s \) and \( t \), we form a new graph \( G' \) by adding a new vertex \( v \), and two new edges \( \{s, v\} \) and \( \{v, t\} \). \( G \) has a Hamiltonian path between \( s \) and \( t \) iff \( G' \) has a 2-factor having at most 1 connected component. If \( G \) has a Hamiltonian path between \( s \) and \( t \), then we can take the edges of that Hamiltonian path and add \( \{s, v\} \) and \( \{v, t\} \) to it to get a 2-factor that has exactly 1 connected component. Conversely, in any 2-factor of \( G' \) having 1 connected component, we will find a cycle containing all the vertices of the graph where the edges \( \{s, v\} \) and \( \{v, t\} \) are present. Deleting these two edges we find a Hamiltonian path between \( s \) and \( t \).

(4)

**ANTIDUNDAS\(\in \text{NP}****: The certificate is similar to the DUNDAS problem, and in this case we check if the 2-factor has at least \( c \) connected components.

The proof of NP-hardness is more complicated. We use the intermediate problem \( \Delta-\text{FACTOR} \): Given a graph \( G = (V, E) \) does \( G \) contain a set of exactly \( |V|/3 \) vertex-disjoint triangles. This is the special case of \( \text{ANTIDUNDAS} \) with \( c = |V|/3 \) and so (1) is in NP by the same argument, and (2) polynomial time reduces to \( \text{ANTIDUNDAS} \) (simply set \( c = |V|/3 \)).

Now we reduce 3CNF SAT to \( \Delta-\text{FACTOR} \). Let \( \phi \) be a 3CNF formula with variables \( x_1, \ldots, x_n \) and clauses \( C_0, \ldots, C_{\ell-1} \). First we form a variable gadget for each variable. For variable \( x_i \) it has vertices \( \{a_{ij}, b_{ij}, t_{ij}, f_{ij} : 0 \leq j < \ell \} \), and edges

\[
\{(a_{ij}, t_{ij}), (a_{ij}, b_{ij}), (a_{i,j+1 \mod \ell}, b_{ij}), (b_{ij}, f_{ij}), (b_{ij}, t_{ij}), (a_{i,j+1 \mod \ell}, f_{ij}) : 0 \leq j < \ell \}
\]

It will be important later that this gadget has exactly two ways to choose exactly \( \ell \) vertex-disjoint triangles. One set leaves all of \( \{t_{ij} : 0 \leq j < \ell \} \) uncovered; the other leaves all of \( \{f_{ij} : 0 \leq j < \ell \} \) uncovered.

Next we build a clause gadget using some vertices from the variable gadgets and some new vertices. For each clause \( C_j \) we add two new vertices \( y_j, z_j \) and the edge between them. Then if \( x_i \) appears in clause \( C_j \),
we add edges between \( t_{ij} \) and both of \( y_j, z_j \); If \( x_i \) appears in clause \( C_j \), we add edges between \( f_{ij} \) and both of \( y_j, z_j \). The 5-vertex clause gadget has three ways to choose a single triangle.

We use \( n\ell \) of the vertices \{\( f_{ij}, t_{ij} \)\} in the triangles on the variable gadgets, and another \( \ell \) of them in triangles on the clause gadgets. So if we stopped here, the most we could hope for is to get \( n\ell + \ell \) vertex-disjoint triangles (on \( 3n\ell + 3\ell \) vertices) but we have \( 4n\ell + 2\ell \) vertices. We have to handle the last \((n-1)\ell \) vertices. To do this add a set \( O \) of \( 2(n-1)\ell \) vertices. Place exactly \( n\ell \) vertex-disjoint edges on the vertices of \( O \). Then add all edges between a vertex in \{\( t_{ij}, b_{ij} : 1 \leq i \leq n, 0 \leq j < \ell \)\} and a vertex of \( O \). (This is a "garbage collection gadget."

This completes the reduction to a graph \( G \), which is constructed in polynomial time. We must check that it is correct. First suppose that \( \phi \) has a satisfying assignment. For each clause, choose one of the literals to be the witness that the clause is true. On the clause gadget, use the triangle involving the witness literal. On the variable gadget for variable \( x_i \), use the \( \ell \) triangles missing \{\( t_{ij} : 0 \leq j < \ell \)\} if \( x_i \) is false, and the \( \ell \) triangles missing \{\( f_{ij} : 0 \leq j < \ell \)\} if \( x_i \) is true. Note that because the witness is a true literal, none of the triangles chosen can involve a triangle chosen from a variable gadget. Now for each of the \((n-1)\ell \) disjoint edges on \( O \), add a (different) vertex from \{\( t_{ij}, b_{ij} : 1 \leq i \leq n, 0 \leq j < \ell \)\} not yet in a triangle to the edge to form a triangle.

In the other direction, suppose that \( G \) has a \( \Delta \)-factor. For every \( 0 \leq j < \ell \), there must be a triangle containing \( y_j \). This triangle must also involve \( z_j \) because the other neighbours of \( y_j \) are not adjacent to each other. Whenever \{\( t_{ij}, y_j, z_j \)\} is a triangle chosen, set \( x_i \) true, and whenever \{\( f_{ij}, y_j, z_j \)\} set \( x_i \) false. We must ensure that we have not set a variable both to true and to false. However, because there are exactly \( 6n\ell \) vertices, \{\( t_{ij}, b_{ij} : 1 \leq i \leq n, 0 \leq j < \ell \)\} contains \( 2n\ell \) vertices, and there are no edges on the latter set, every triangle must contain exactly one of \{\( t_{ij}, b_{ij} : 1 \leq i \leq n, 0 \leq j < \ell \)\}. But if a variable gadget has both a \( t \) and an \( f \) vertex removed, it cannot contain \( \ell \) vertex-disjoint triangles, and so we have a contradiction. Thus the chosen assignment is consistent and satisfies \( \phi \). (One last note: It may happen that a variable is never used as a witness. If that happens, assign its truth value arbitrarily.)

Therefore \( \Delta \)-FACTOR is NP-complete, and therefore ANTIDUNDAS is NP-complete.

(5)  

No. The argument is not sufficient. It only shows that COUNTSAT is NP-hard. In fact, it is not clear whether COUNTSAT ∈ NP. An obvious certificate for proving the membership would be \( k \) different satisfying assignments. But the number \( k \) which is part of the input may be specified in \( O(\log k) \) bits in any alphabet other than the unary alphabet. That makes the size of the certificate exponential in the size of the input.

So can we repair the argument? The answer is “We don’t know.”