CSE 431 Theory of Computation | Spring 2014

Lecture 15: May 20

Lecturer: James R. Lee | *Scribe: Amit Burstein*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

15.1 Planar Graph Coloring

Recall that $PLANAR\text{-}3\text{-}COL = \{\langle G \rangle : G \text{ is a planar 3-colorable graph }\}$. We want to show that this language is NP-complete using the following gadget:

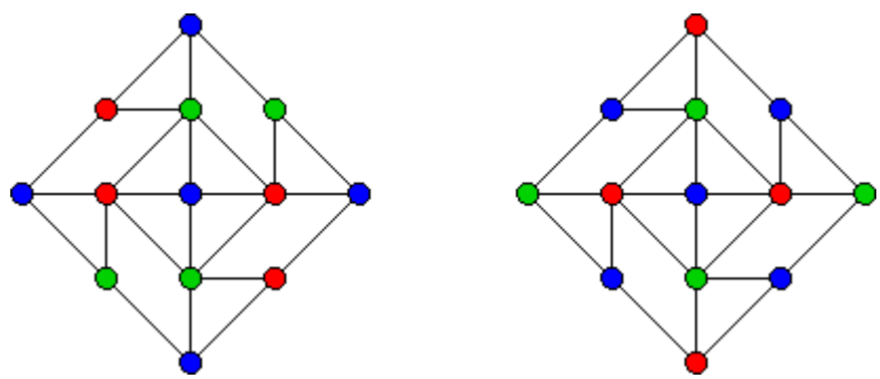

Figure 15.1: Uncrossing gadget

Exercise: *PLANAR-3-COL* is *NP*-complete

Proof: It is trivial to see that a certificate of this problem could be verified in polynomial time by just checking the graph's 3-coloring, so $PLANAR\text{-}3\text{-}COL \in NP$.

To complete this proof, we will give the following reduction: $3\text{-}COL \leq_p PLANAR\text{-}3\text{-}COL$, or less formally, we will construct a new graph $\hat{G}$ from the input graph G such that G is 3-colorable $\iff$ $\hat{G}$ is planar 3-colorable.

To construct $\hat{G}$, we replace all edge crossings in G with the above gadget. This gadget has the key properties that (1) every valid 3-coloring of it has opposite corners the same color and (2) any such coloring of the corners extends to a 3-coloring of the entire gadget. These properties can be shown by enumerating the gadget's possible 3-colorings.

If an edge in G is crossed by multiple other edges, the gadgets that replace those crossings need to be linked together at the edges. This propagates the fact that the nodes at either end of the edge must be different colors.

It is easy to see that $\hat{G}$ is planar 3-colorable, and it also easy to see that removing gadgets from such a graph gives G. This reduction runs in polynomial time, and thus *PLANAR-3-COL* is *NP-complete*.

■

Lemma 15.1 *3-COL* $\leq_p$ *4-COL*

Proof: A simple polynomial time reduction is to add one node to the input graph G which has edges to all other nodes and had the fourth color. ■

Lemma 15.2 *4-COL* $\leq_p$ *3-COL*

Proof: This reduction exists because *4-COL* $\in$ *NP* and *3-COL* is *NP*-complete. You could formulate this reduction by chaining together multiple reductions:

1. *4-COL* $\leq_p$ *SAT* (use a tableau)
2. *SAT* $\leq_p$ *3-SAT*
3. *3-SAT* $\leq_p$ *3-COL*

■

15.2 Space Complexity

For a deterministic TM M, its space complexity is the function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n)$ is the maximum number of tape cells M uses on any input of length n.

For a nondeterministic TM N, $f(n)$ is the maximum number of tape cells N uses on any computation path on any input of length n.

Definition 15.3
SPACE(f(n)) = $\{$*L : L is a language decided by an O(f(n)) space deterministic TM* $\}$
NSPACE(f(n)) = $\{$*L : L is a language decided by an O(f(n)) space nondeterministic TM* $\}$

Space appears to be more powerful that time because space can be reused whereas, time cannot. For example, consider the space complextity of *SAT*. An algorithm that iterates over every possible assignment to ϕ and checks if it is satisfied can reuse the tape cells of the TM on each iteration. This algorithm runs in $O(n)$ space where n is the number of variables in ϕ.

Definition 15.4
PSPACE = $\bigcup_k SPACE(n^k)$
NPSPACE = $\bigcup_k NSPACE(n^k)$

15.2.1 Savitch's Theorem

Whereas whether $P = NP$ is still unknown to us, Savitch's Theorem can be used to show us that *PSPACE* = *NPSPACE*.

Theorem 15.5 *For any $f(n) \geq n$, NSPACE($f(n)$) $\subseteq$SPACE($f^2(n)$)*

Proof: To complete this proof, we want to simulate an $f(n)$ space nondeterministic TM with a deterministic TM. We define the procedure $CANYIELD(c_1, c_2, t)$ that determines if a nondeterministic TM can go from configuration c_1 to c_2 in t steps in $f(n)$ space. We can take c_1 to be the TM's start configuration, c_2 to be its accepting configuration, and t to be the maximum number of steps the TM could use ($c^{f(n)}$).

$CANYIELD(c_1, c_2, t)$:

if $t = 0$
  accept only if $c_1 = c_2$
if $t = 1$
  accept if $c_1 \to c_2$ in one step
else
  for all configurations c_m using $kf(n)$ space
    call $CANYIELD(c_1, c_m, t/2)$
    call $CANYIELD(c_m, c_2, t/2)$
    if both accept, *accept*
  reject

Since *CANYIELD* calls itself recursively and uses c_1, c_2, c_m, and t for each call, $O(f(n))$ stack space is needed for each level of recursion. Since each level divides t in half (t started at $c^{f(n)}$), the stack has $O(log(c^{f(n)})) = O(f(n))$ depth. Altogether, the total space used is thus $O(f^2(n))$.

We will complete this proof in the next lecture.

■