Homework 8 Solutions, CS 132, Winter 2005

March 10, 2005

14.11 *Show that both of these decision problems are in $\mathcal{P}$.*

a. *DNF-Satisfiability: Given a DNF formula is it satisfiable.*

A disjunction is the boolean OR of its clauses, and so it is satisfiable if *any* of its clauses can be satisfied. A clause is the boolean AND of some number of literal terms. Any clause is satisfiable unless it contains both some literal x and the negation of that literal $\overline{x}$. Thus an algorithm to determine DNF-Satisfiability would loop through the clauses and check if there is a clause which is satisfiable. This is clearly in $\mathcal{P}$.

b. *CNF-Tautology: Given a CNF formula, is it a tautology?*

There are two methods of proving this:

i. A CNF formula is the boolean AND of its clauses. Thus it is a tautology if and only if every clause is a tautology. A clause is the boolean OR of some number of literals. Thus a clause is only a tautology if and only if it has some literal x and the negation of that literal $\overline{x}$. This can easily be checked by scanning through the clauses, and so CNF-Tautology is in $\mathcal{P}$.

ii. A more clever method of proving this is to note that by DeMorgan's law the negation of a DNF formula is a CNF formula. Further, note that if X is a DNF tautology then its negation is a CNF contradiction, i.e. it is unsatisfiable. Thus we can solve this problem by negating the input and using our solution to the previous problem.

14.16 *Consider the following algorithm to solve the vertex cover problem. First, we generate all size-k subsets of the vertices. There are $O(n^k$ of them. Then we check whether any of the resulting subgraphs is complete. Why is this not a polynomial-time algorithm?*

For it to be a polynomial time algorithm, it must have runtime which is polynomial for any input. However k and n *both* depend on the input, so n^k is not a polynomial. For example, consider the input $(G = (V, E), n)$ where $|V| = n$ in this case the algorithm generates $O(n^n)$ sets.

14.25 *Show that the following decision problem is $\mathcal{NP}$-complete: Given a graph G in which every vertex has even degree, and an integer k, does G have a vertex cover with k vertices?*

VC-EVEN is a special case of **VC** so it follows trivially that **VC-EVEN** $\leq_P$ **VC** and hence **VC-EVEN** is in $\mathcal{NP}$.

Now we show that **VC-EVEN** is $\mathcal{NP}$-hard by showing that **VC** $\leq_P$ **VC-EVEN**. To do this, we must describe a reduction f such that given (G, k), an instance of **VC**, $f(G, k) = (G', k')$ such that every vertex in G' has even degree and such that (G', k') is a yes-instance of **VC-EVEN** if and only if (G, k) is a yes-instance of **VC**.

$G' = (V', E')$ will be constructed as follows: if $G = (V, E)$, then

$$V' = V \cup \{v_1, v_2, v_3\}, \text{ and}$$

$$E' = E \cup \{(v_1, v_2), (v_2, v_3), (v_1, v_3)\} \cup \{(v, v_1) | v \in V \text{ and } v \text{ has odd degree}\}$$

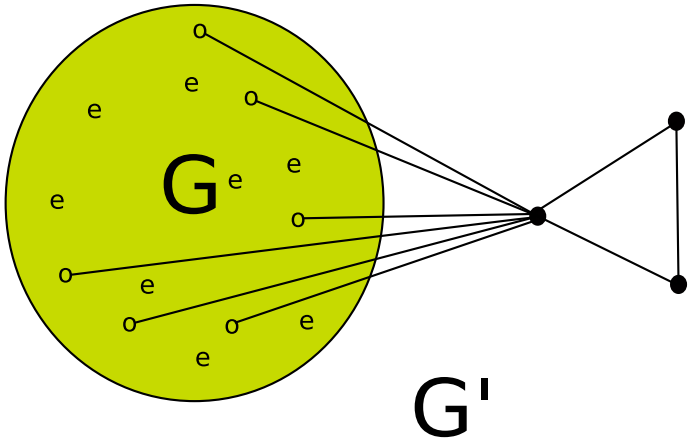

Figure 1: The Reduction from **VC** to **VC-EVEN**. "o" denotes a vertex with odd degree, and "e" a vertex with even degree.

And, finally, $k = k' + 2$.

The odd-index vertices can be found in a single traversal of the graph, so f clearly can be computed in polynomial time.

Now we must show that if and only if G and k are a yes-instance of **VC** then G' and $k'$are a yes-instance of **VC-EVEN**. We begin with the "if": clearly if there is a vertex cover for G, $C \subseteq V$ such that $|C| = k$ then $C \cup v_1, v_3$ is a vertex cover for G' of size $k' = k + 2$. Now the "only-if". Assume that G' has a vertex cover of cardinality k'. At least two of the verticies $v1, v2, v3$ must be in the vertex cover of G' for that triangle to be covered, therefore, since all edges in G' are covered by $k+2$ vertices, the remaining edges must be covered by k. These remaining edges are exactly those in G.

We must also show that every vertex in G' has even degree.This follows directly from the lemma below.

Lemma: For any graph $G = (V, E)$, the number of odd-degree vertices is even.

Proof: The sum of the degree of all vertices in a graph with m edges is $2m$ since each edge contributes 1 to the degree of exactly two vertices. Call the set of odd-degree vertices $V_o \subseteq V$, then

$$\sum_{v \in V} degree(v) = \sum_{v \in V_o} degree(v) + \sum_{v \notin V_o} degree(v) = 2m$$

Since both $2m$ and the sum of the even degree vertices are even, so must the sum of the odd degree vertices. Since their degree is odd if the sum of their degree must always be even then their number must be even also.

14.30 ***PARTITION*** *is the following problem: given a sequence* $A = a_1, a_2, ..., a_n$ *of integers, is there a subset* J *of* $\{1, 2, ..., n\}$ *so that* $\sum_{i \in J} a_i = \sum_{i \notin J} a_i$*? Show this problem to be* $\mathcal{NP}$*-complete by showing* ***SUBSETSUM*** $\leq_P$ ***PARTITION***.

First note that if we nondeterministically guess a subset J we can easily check the equality of the sums; therefore **PARTITION** is in $\mathcal{NP}$.

Now we will show that it is $\mathcal{NP}$-hard by giving the above reduction. To carry out this reduction we must construct a function $f \in \mathcal{P}$ such that iff (A, K) is an instance of **SUBSETSUM** then $f(A, k) = (A')$ is an instance of **PARTITION**. If $A = a_1, a_2, ..., a_n$ is the sequence of integers in the input to **SUBSETSUM**, and $s = \sum_{i=0}^{n} a_i$, we will let A', the sequence used as input for **PARTITION**, be $A' = a_1, a_2, ..., a_n, k + 1, s - k + 1$.

The computation of f requires only $O(n)$ time—the time necessary to sum the integers in A.

If A, k is a yes-instance of **SubsetSum** such that the sum of A is s, then there is some set of integers I such that $\sum_{i \in I} a_i = k$. There must then be a partition in A', since $k + 1 + \sum_{j \notin I} a_j = \sum_{i \in I} a_i + (s - k + 1) = s + 1$.

Assume A' has a partition, we will show that A must have then had a subset that sums to k. Clearly $k + 1$ and $s - k + 1$ cannot be on the same side of the partition, as their sum is greater than that of the remaining elements. Therefore the partition must be of the form

$$(s - k + 1) + \sum_{i \in I} a_i = (k + 1) + \sum_{j \notin I} a_j$$

where I is some set of integers between 1 and n. Clearly the elements $s - k + 1$ and $k + 1$ can't be on the same side of the partition, as they are larger than the remaining elements. Since by definition s is the sum of the first n elements,

$$(s - k + 1) + \sum_{i \in I} a_i = (k + 1) + (s - \sum_{i \in I} a_i)$$

and so, with some simplification, we get

$$k = \sum_{i \in I} a_i.$$

Therefore **SUBSETSUM** $\leq$ **PARTITION**, and since as we showed above, **PARTITION** is in $\mathcal{NP}$, it follows that **PARTITION** is $\mathcal{NP}$-complete.

14.22 Extra Credit: *Show that the 2-Satisfiability problem is in $\mathcal{P}$.*

First we will describe a method for creating a graph which represents instances of 2-Sat, then we will give a polynomial time algorithm that solves 2-Sat using this graph.

Let g be the function which creates the graph, and I an arbitrary instance of 2-Sat having n literals, then

$$g(I) = G = (V, E) = (\{x_1, ..., x_n, \neg x_1, ..., \neg x_n\}, \{(x_i, x_j) | \text{either } (\neg x_i \vee x_j) \text{ or } (\neg x_j \vee x_i) \text{ is a clause in } I\})$$

The edges in the graph are thus the implications formed by each clause.

Lemma: I is satisfiable if and only if there is no $x_i \in V$ such that there is a path from x_i to $\neg x_i$ and from $\neg x_i$ to x_i in $g(I)$.

Proof: If I is unsatisfiable then for any truth assignment there is some clause for which the truth assignment is $(F \vee F)$, or in our formulation, $(T \Rightarrow F)$. Note also the symmetry that if we have an edge (α, β) then we have also $(\neg\beta, \neg\alpha)$.

First we show that if there is a path $x \rightsquigarrow \neg x$ and a path $\neg x \rightsquigarrow x$ in $g(I)$ then I is unsatisfiable. If x is true, then by the transitivity of $\Rightarrow$ if there is a path $x \Rightarrow ... \Rightarrow \neg x_i$ then there must be some literal on that path that is false, which means $T \Rightarrow F$ must have appeared in I; if x is false then we have the same occurence in $\neg x \Rightarrow ... \Rightarrow x$. Since there is at least one unsatisfied clause, I is unsatisfiable.

Now suppose $g(I)$ contains no such paths, we will show that I has a satisfying truth assignment. Begin with an arbitrary vertex α such that $(\alpha, \neg\alpha)$ is not an edge in the graph. Give all β such that β is reachable from α the assignment true, and give $\neg\beta$ the assignment false. This is always legitimate: if there was a path $\alpha \rightsquigarrow \beta$ and $\alpha \rightsquigarrow \neg\beta$ then by symmetry there would be a path $\neg\alpha \rightsquigarrow \neg\beta$ which contradicts our assumption. Therefore we can create a satisfying assignment for every clause which contains a literal reachable from α. Repeating this process for every unassinged literal in I creates a satisfying assingment.

Now we can show that 2-Sat is in $\mathcal{P}$. First we use DFS to find the strongly-connected components of $g(I)$ (CLR gives a linear algorithm to do this with only two depth-first searches), then we check if

any literal appears in the same strongly-connected component as its negetion. A strongly-connected component is a set of vertices such that for any two vertices in the set there is a path between them. Therefore the correctness of the algorithm follows directly from the above lemma.