Computational Complexity Theory, Fall 2010 September 24

Lecture 10: Counting complexity

Lecturer: Kristoffer Arnsfelt Hansen Scribe: Camilla Gravgaard

Padding technique

Theorem 1. *If $NP = PSPACE$ then $NEXP = EXPSPACE$*

Proof. Let $L \in EXPSPACE$, and let M be a deterministic TM deciding L, in space 2^{n^k}. Now define: L_{pad}=$\{x\#^{2^{|x|^k}-|x|} : x \in L\}$. (That is, an input x of is padded to an input of length $2^{|x|^k}$.)
Claim 1. $L_{pad} \in PSPACE$

Proof of claim. We can use the following algorithem to prove that L_{pad} is inside $PSPACE$.

1. Check if input y is of the form $x\#^{2^{|x|^k}-|x|}$
2. Remove #'s, and run M on x.

Space analysis: $|x|^k + 2^{|x|^k}$=$O(|y|)$ □

By assumption $L_{pad} \in NP$. So, let M' be a nondeterministic TM accepting L_{pad} in time $n^{k'}$.
Claim 2. $L \in NEXP$

Proof of claim. Algorithm on input x, write $2^{|x|^k} - |x|$ # symbols after x, and run M'.
Time usage: $2^{|x|^k}$+$(2^{|x|^k})^{k'}$=$2^{|x|^k}$+$(2^{k'(|x|^k)})$. □

□

Counting complexity

Example: Given a boolean formula φ. Compute the number of satysifing assinments in φ.
Define: $\#P$ = class of functions $f : \Sigma^* \to \mathbb{N}$ such that there is a polynomial p and $R \in P$ such that $f(x) = \left|\{y \in \{0,1\}^{p(|x|)} :< x, y >\in R\}\right|$

Also, define FP =class of functions $f : \Sigma^* \to \mathbb{N}$ such that the function $x \mapsto bin(f(x))$ is P-computable.

Let $g : \Sigma^* \to \mathbb{N}$ be a function. Define further FP^g = class of functons $f : \Sigma^* \to \mathbb{N}$ such that the function $x \mapsto bin(f(x))$ is in P^{L_g}.

Definition 2. f is #P-complete if:

- $f \in \#P$
- for any $g \in \#P$, $g \in FP^f$

Theorem 3. *#SAT and #3SAT are #P-complete.*

Proof. To prove this we can use the following reductions, already seen in the proof of Cook's theorem.

- $L \leq CircuitSAT$, for any $L \in NP$
- $CircuitSAT \leq 3SAT$

The crucial property of these reductions that allow us to draw our conclusion is that they exactly *preserve* the number of witnesses. □

Definition 4 (Determinant and Permanent)**.** Let A be an $n \times n$ matrix. Then the determinant is given by

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

and the permanent is given by

$$Perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

Theorem 5 (Valiant's theorem)**.** *The Permanent function over 0/1 matrices is #P complete.*

Before we do the proof, here is an interpretation.

For a 0/1 matrix A: Define a bipartite graph G on $2n$ vertices by: $(i,j) \in E \Leftrightarrow A_{i,j} = 1$

Then $Perm(A)$ is the number of perfect matchings in G.

For a general graph: Define a directed graph D on n vertices with weights: $(i,j) \in E \leftrightarrow A_{i,j \neq 0}$, Weight of (i,j), $w(i,j) = A_(i,j)$.

Any permutation σ is a product of cycles. These coresponds to cycles in the graph D, and hence σ correspond to a *cycle-cover* in D. Hence we get:

$Perm(A) = \sum_{\text{Cyclecovers of}D}$ Product of weights in the cycle-cover

Now we are ready for the proof of Valiant's theorem, where we will use the interpretation in terms of cycle covers.

Proof of Valiant's Theorem. This amazing reduction is from the #3SAT function.

Given a 3CNF φ with m clauses and n variables, we will construct an integer matrix A such that $Perm(A) = 4^{3m} \cdot$ #sat.assignment of φ

Idea of the reduction: A cycle gives an assignment to the variabls, and assignment to literals in the clauses, that satisfyes the clauses. Using an XOR gadget we will cancel contribution from cycle covers that are not consistent.

For each variable, we will have a variable-gadget, see figure 1. Bold edges will be connected to clause-gadgets using XOR-gadgets, corresponding to whether the literal in the clause is positive or negative. The truth value of the variable is detmined by whether the top or the bottom nodes are just covered by the self loops, or they are covered by a cycle through the bold edges.

For each clause, we will have a clause gadget, see figure 2. Bold edges connect to variable-gadgets as described above. The important property of this gadget is that, since the center node

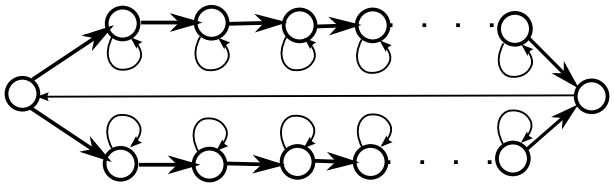

Figure 1: Variable gadget

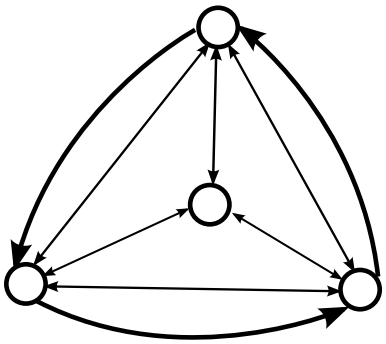

Figure 2: Clause gadget

must be in a cycle there must be at least one bold edge that is not traversed by a cycle (hence ensuring that one of the literals must be true).

The most interesting gadget is the XOR-gadget, see figure 3. This gadget is connected between edges (U, U') and (V, V'), and will ensure that if a cover traverses 0 or 2 of the edges, then the contribution from this cycle cover is cancelled out (i.e the contribution is multiplied by 0). If on the other hand a cycle cover traverses exactly one of the two edgrs, then the contribution will be multiplied by 4.

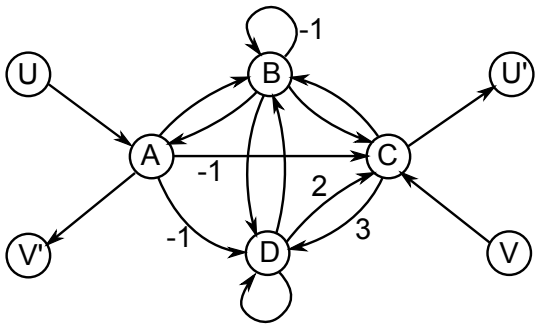

Figure 3: XOR gadget

We now nalyse the multipler based on which of the edges UA,AV',CU',VC are in the global cycle cover. We list the cycles in the graph using cycle notation (e.g (AB) is the 2-cycle through A and B). For an outside cycle we use brackets instead (e.g $< ABC >$ denotes a cycle that enters the gadget in node A, goes through B and exits the gadget in in C).

1. No edges are used:

(AB)(CD): 6
(ADCB) : -2
(ACDB) : -3
(ACB)(D): -1
$Sum = 0$

2. Exactly UA and AV':
   (BCD) : 3
   (BDC) : 2
   (BC)(D) : 1
   (B)(CD) : -6
   $Sum = 0$

3. Exactly VC and CU'
   (ADB) : -1
   (AB)(D) : 1
   $Sum = 0$

4. All edges are used (i.e. UA, AV', VC, CU')
   (BD) : 1
   (B)(D) : -1
   $Sum = 0$

   We thus see that if we use the XOR gadget in any of these ways, we get no contributation.

5. Exactly UA and CU':
   $< AC >$(BD) : -1
   $< AC >$ (B)(D) : 1
   $< ABC >$ (D) : 1
   $< ADC >$ (B) : 2
   $< ADBC >$: -1
   $< ABDC >$: 2
   $Sum = 4$

6. Exactly VC and AV':
   $< CBA >$(D) : 1
   $< CDBA >$: 3
   $Sum = 4$

The last two ways of using the XOR gadgat gives ud a contributation of 4, therefor we get a factor 4 ind the formula from the beginning of the proof. Connecting the variable and clause-gadgets using XOR-gadgets uses exactly one XOR-gadget for each literal in the formula (i.e $3m$). Thus every satisfying assigment contributes 4^{3m} to $Perm(A)$.

The next part of the proof deals with reducing this to the case of 0/1 matrices. Although the above matrix just uses weights from $\{-1, 0, 1, 2, 3\}$, we describe how to do the reduction for any integer matrix.

1. First we make all weights powers of 2. Consider the binary representation of a weight w. Replace the edge with parallel edges for each 1 in the binary representation, with the corresponding power of 2 weight.

2. Next we make all weights belong to $\{-1, 0, 1\}$. A weight 2^k (or -2^k) may be replaced by a gadget, see figure 4, that replaces an edge by a graph that has 2^k different paths (and unused nodes can be in a self loop). (In case of weight -2^k we may make the 2 first edges have weight -1).

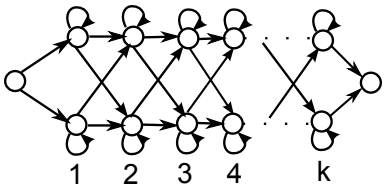

Figure 4: Reduction from powers of 2

3. Finally make all weights belong to $\{0, 1\}$. Let A be an $n \times n$ matrix with entries in $\{-1, 0, 1\}$ Now, $Perm(A) \in \{-n!, ..., n!\}$

   Define: $A^{'}$ and replace $-1's$ from A with $2n! + 1$ and note that $2n! + 1 \equiv -1 \pmod{(2n! + 2)}$.

   Then $Perm(A^{'}) = Perm(A) \pmod{(2n! + 2)}$, and if we can thus determine $Perm(A)$ from $Perm(A^{'})$. However now weights are not $\{0, 1\}$ as promised, but they are *positive*, and we just have to repeat step 1 and 2 to get 0/1 entries. QED.

□