

Complexity & Advanced Algorithms

Siddharth Bhat

Contents

1	NLogSpace-completeness	5
1.1	Co-NLogSpace	5
1.1.1	Solving $\overline{\text{PATH}}$ in NLogSpace	5
1.2	Oracles	6
1.2.1	P^{poly}	7

Chapter 1

NLogSpace-completeness

1.1 Co-NLogSpace

$L \in \text{Co-NLogSpace} \equiv L^c \in \text{NLogSpace}$. That is, complement the language L . if L^c is in NLogSpace , then $L \in \text{Co-NLogSpace}$.

We intuitively believe that $\text{NP} \neq \text{Co-NP}$. However, we can show that $\text{NLogSpace} = \text{Co-NLogSpace}$.

$$\begin{aligned}\text{PATH} &= \{\langle G, u, v \rangle \mid \text{exists path between vertices } (u, v)\} \\ \overline{\text{PATH}} &= \{\langle G, u, v \rangle \mid \text{no path between vertices } (u, v)\}\end{aligned}$$

We assume that $\overline{\text{PATH}}$ is Co-NLogSpace-Complete.

If we show that $\overline{\text{PATH}}$ is in NLogSpace , then every problem in Co-NLogSpace will be in NLogSpace .

1.1.1 Solving $\overline{\text{PATH}}$ in NLogSpace

$$\begin{aligned}V_R &\equiv \{\text{set of vertices reachable from } u\} \\ V_{NR} &\equiv \{\text{set of vertices **not** reachable from } u\}\end{aligned}$$

Sid confusion, why can't we use PATH as a subroutine: When we have an NDTM, we cannot *observe that the NDTM returns a 0*. We can *observe if an NDTM succeeds*, but there are weird paths and exponential number of paths where the NDTM does not return a 0? But if this is true, then how is PATH NLogSpace -complete? I am very confused.

To represent V_R and V_{NR} , we use 1 bit per vertex (since V_R and V_{NR} are disjoint), so total space is V .

Assume we know $|V_R|$. In this case, we can check whether v is unreachable from u — Enumerate all vertices. If they are reachable from u , bump up a counter. If we don't hit v till the counter gets to $|V_R|$, then what we know that is v is unreachable.

However, if v were reachable from u , then as we enumerate, we would find v as we were going through all vertices (we would not hit V_R unless we visit v).

This is important, because in an NDTM, if *any* of the paths accept, then we accept.

$$V_R = \cup_i V_R(i)$$

$$V_R(0) = \{u\}$$

to compute $cur \in? V_R(i+1)$, first **recompute** that $pred \in V_R(i)$, and then check that $(cur, pred) \in E(G)$. We cannot **store** $V_R(i)$, since we don't have enough space.

eventually we will reach $V_R(|V|)$, where we stop.

We can compute $|V_R| = \sum_i |V_R(i)|$. We compute $|V_R(i)|$ by checking over each vertex it's membership into $V_R(i)$. And if it does, we bump up our counter.

Reference: Read Sipser-Chapter 8

```
def belongs(G, i, startv, endv, curv):
    """Check if curv belongs to V_R(i)"""
    if i == 1:
        return startv == curv
    else:
        # log(V)
        for pred in G.vertices:
            # This can use a modified version of PATH that stores lengths?
            if small_belongs(G, i - 1, startv, endv, pred):
                if isneighbour(pred, curv):
                    return True

        return False

def countcard(G, startv, endv):
    """Count the cardinality of V_R"""
    card = 0
    # log(V)
    for i in len(G.vertices):
        # this is also log(V)
        for curv in G.vertices:
            if small_belongs(G, i, startv, endv, curv):
                card += 1
    return card
```

1.2 Oracles

For all inputs w of length $|w| = n$, there exists a **single** advice (a_n is allowed to be a single string that is polynomial in n). So, $a : \mathbb{N} \rightarrow \Sigma^*$, and the advice of a given input w is $a(|w|)$.

1.2.1 P^{poly}

$L \in \text{P}^{\text{poly}}$ if there is a polynomial time turing machine M which takes two inputs — a string $x \in \Sigma^*$, and an advice $a_n \in \Sigma^*$, such that for all inputs w such that $|w| = n$, then there exists a polynomial $p(n)$ with $|a_n| \leq p(|w|)$.

We force it to be polynomial in the word-length, because things like a lookup table take exponential space in the word-length (number of strings of length n is 2^n).

We can see that the advice is somewhat "hardwired" into the machine given the input length (since $a : \mathbb{N} \rightarrow \Sigma^*$). So, we have a sequence of machines $M_i : \mathbb{N} \rightarrow \{\text{Turing machines}\}$, and we instantiate the machine $M_{|w|}$ to check if $|w| \in L$.

NP is allowed to have a *varying witness*, while P^{poly} will have the *same* advice.