INFR11102: Computational Complexity 03/10/2019

Lecture 6: Savitch's Theorem

Lecturer: Heng Guo

1 Savitch's theorem

In [Sav70], Savitch proves the following theorem.

Theorem 1 ([Sav70])**.** *Let $S(n) \geq \log n$ be a space-constructible function. Then*

$$\texttt{NSpace}[S(n)] \subseteq \texttt{DSpace}[(S(n))^2].$$

The crux of Savitch's theorem is a space efficient algorithm for the following problem.

Name: Conn

Input: A directed graph $G = (V, E)$ with two vertices s and t.

Output: Is there a path from s to t in G?

We may assume some simple representation of the input, like the adjacency matrix, which has size $O(|V|^2)$.

This is one of the most basic problems in an algorithm class. We all know that we can solve it by, say, DFS or BFS. However, standard DFS or BFS would record 1 bit for each vertex whether it has been visited, thus requiring $\Omega(n)$ space where $n = |V|$.[1] Savitch, instead, gives a space-efficient version.

Theorem 2 ([Sav70])**.** Conn $\in \texttt{DSpace}[(\log n)^2]$.

So we know Conn $\in \texttt{DTime}[n]$ by BFS/DFS, and Conn $\in \texttt{DSpace}[(\log n)^2]$ by Theorem 2, but we do not know whether the two can be achieved at the same time.

Savitch's algorithm is a recursive procedure Reach(u, v, ℓ) given in Algorithm 1, where $\mathbb{1}(A)$ is the indicator function of an event A. Namely, $\mathbb{1}(A) = 1$ if A is true, and $= 0$ otherwise. Clearly Conn can be solved by Reach(s, t, n) where $n = |V|$.

The basic idea is the following: we enumerate all possible "middle point" w of u and v, and recursively determine whether u and v are both connected to w at most $\ell/2$ steps. Such w exists, if and only if, u and v are connected by a path of length at most ℓ. This shows the correctness of Algorithm 1.

We claim the space complexity of Algorithm 1 is $O((\log n)^2)$. This is because, the recursion depth of Algorithm 1 is at most $O(\log n)$. At each level of the recursion, we need to

[1] Technically, the input size of a graph G can be quadratic in the number of vertices. However, since we are mostly concerned with log-spaces or polynomial time, it does not make a difference, up to constants.

Algorithm 1 $\textsc{Reach}(u, v, \ell)$

Input: Graph $G = (V, E)$, two vertices u and v, and an integer ℓ.
Output: Is there a path from u to v of length at most ℓ?
if $\ell = 1$, **then**
  return $\mathbb{1}(u = v \text{ or } (u, v) \in E)$.
end if
for $w \in V$, **do**
  if $\textsc{Reach}(u, w, \lfloor \ell/2 \rfloor) = 1$ and $\textsc{Reach}(w, v, \lceil \ell/2 \rceil) = 1$, **then**
    return 1.
  end if
end for
return 0.

store information about u, v, w, and ℓ, each of which requires at most $\log n$ bits. Thus, in total, we need to store $O(\log n)$ bits at each of the $O(\log n)$ recursion levels, resulting in a total space complexity $O((\log n)^2)$ of Algorithm 1.

In general, recursive procedures are usually space-efficient. The total space is $O(d \cdot s)$, where d is the recursion depth and s is the number of bits required for each level. Of course one can easily unravel the recursion using, for example, stacks, but the recursive version is often easier to describe and analyze.

To prove Theorem 1, we want to apply Algorithm 1 to the configuration graph of the NTM. However, to explicitly construct the configuration graph would require too much space. The saving grace is that Algorithm 1 only requires to know local structure of the configuration graph.

Proof of Theorem 1. For any $L \in \mathtt{NSpace}[S(n)]$, suppose it is computed by an NTM N with $S(n)$ space. For an input x where $n = |x|$, we invoke Algorithm 1 with $G = G_{N,x}$, the configuration graph, u being the initial configuration, v being the accepting configuration, and $\ell = 2^{cS(n)}$, the maximum number of total configurations, where c is a constant depending only on N. Note that the accepting configuration may not be unique in $G_{N,x}$, but this is not an issue since we can simply identify all of them. There are no outgoing arcs from accepting configurations anyways.

The only subtlety is that we cannot afford to construct the whole $G_{N,x}$. In the execution of Algorithm 1, the enumeration step will go over all possible configurations up to space $S(n)$. If there is a query whether $u = v$ or $(u, v) \in E$, we can easily determine it by checking the transition function of N. This takes at most constant bits.

Since the size of $G_{N,x}$ is at most $\ell = 2^{cS(n)}$, the total space complexity of this algorithm is $(\log \ell)^2 = O((S(n))^2)$.

Space constructibility is required since we need to know an upper bound of the space used in order to enumerate all possible configurations. □

An immediate consequence of Theorem 1 is that $\mathtt{PSpace} = \mathtt{NPSpace}$. However, it is not clear that whether $\mathtt{NL} = \mathtt{L}$. Theorem 1 only implies that $\mathtt{NL} \subseteq \mathtt{DSpace}[(\log n)^2]$.

2 Log-space computation

It is clear that $\textsc{Conn} \in \texttt{NL}$. We will show that $\textsc{Conn}$, in fact, is a $\texttt{NL}$-complete problem, in the sense that if we can solve $\textsc{Conn}$ in $O(\log n)$ space, then we can solve any problem in $\texttt{NL}$ in $O(\log n)$ space. In other words, if $\textsc{Conn} \in \texttt{L}$, then $\texttt{L} = \texttt{NL}$.

Definition 1. *A language B is* log-space reducible *to a language A, denoted $B \leq_\ell A$, if, there is a TM M that uses $O(\log n)$ space and for any $x \in B \Leftrightarrow M(x) \in A$.*

A language A is $\texttt{NL}$-hard *if for every language $B \in \texttt{NL}$, $B \leq_\ell A$.*

A language A is $\texttt{NL}$-complete *if it is $\texttt{NL}$-hard and $A \in \texttt{NL}$.*

Remark. In Definition 1, we use $M(x)$ to denote the output of M on input x, which, in this case, is not necessarily 0/1. Recall that in space complexity, we only count the size of the work tape. The output of M does not need to be of logarithmic size.

The intuition for the notation $B \leq_\ell A$ is that B is easier than A — if we can solve A, we can solve B.

Theorem 3. $\textsc{Conn}$ *is* $\texttt{NL}$*-complete.*

Proof. It is easy to see that $\textsc{Conn}$ is in $\texttt{NL}$, since we can simply guess the path.

For a language $B \in \texttt{NL}$, let $B = L(N)$ for an NTM N. Given an input x, the reduction simply output the configuration graph $G_{N,x}$, along with its starting and accepting configurations. The space used to construct $G_{N,x}$ is $O(\log n)$. For example, we can first write down all possible configurations, then for each configuration, go through all other configurations to see what are its neighbours. The space needed is proportional to the size of a single configuration. □

Remark (Bibliographic). Savitch's Theorem can be found in [AB09, Theorem 4.14] and [Pap94, Theorem 7.5]. Relevant chapters are [AB09, Chapter 4] and [Pap94, Chapter 7.3].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.