Complexity Theory

Lecture 9: Space Complexity

Tom Gur

Sublinear Complexity!

Space Complexity

SPACE(f) is the set of languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

Counting only work space.

NSPACE(f) is the class of languages accepted by a *nondeterministic* Turing machine using at most $O(f(n))$ work space.

As we are only counting work space, it makes sense to consider bounding functions f that are less than linear.

Space Complexity Zoo

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

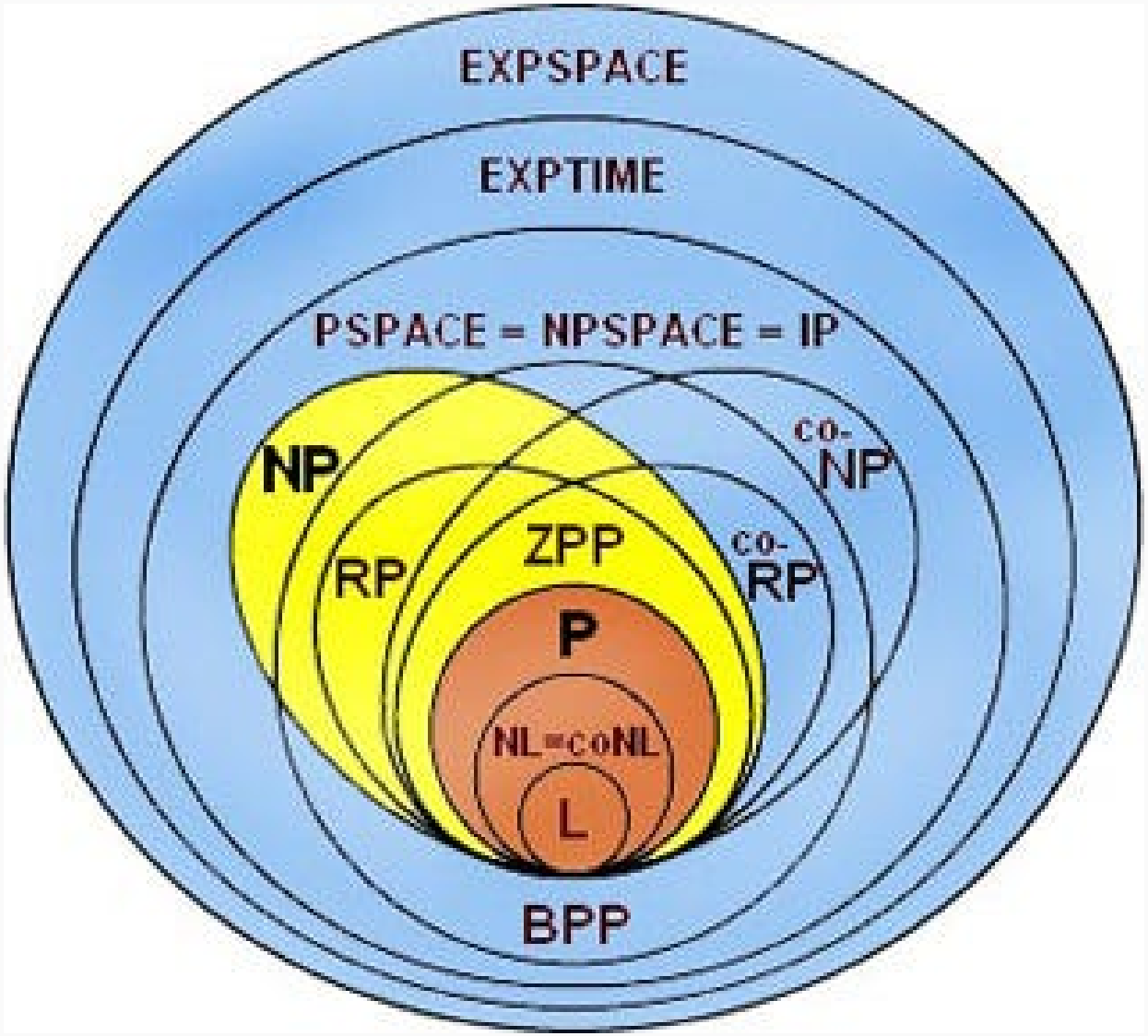
$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define:

co-NL – the languages whose complements are in NL.

co-NPSPACE – the languages whose complements are in NPSPACE.

Space Complexity Zoo



ST-Conn and NL

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t. Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

Complexity: $O(n^2)$ time, $O(n)$ space.

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   - 2.1 if $i = t$ then accept, else
     guess an index j ($\log n$ bits) and write it on the work space.
   - 2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

When in vertex i, the algorithm tries all possible indices j in parallel.

For edges (i,j), the computation can continue.

If there is a path from s to s, there will be a computation that visits all the nodes on that path.

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

We can prove that st-Conn is in NL as follows:

- Start with an NL machine.
- Construct its configuration graph.
- Run an st-Conn algorithm and accept iff it accepted.

Configuration Graph

Define the *configuration graph* of M, x to be the graph whose nodes are the possible configurations, and there is an edge from i to j if, and only if, $i \to_M j$.

Then, M accepts x if, and only if, some accepting configuration is reachable from the starting configuration $(s, \triangleright, x, \triangleright, \varepsilon)$ in the configuration graph of M, x.

L vs NL

L vs NL

The problem st-Conn is NL-complete. Can we solve it deterministically?

Theorem (Savitch's Theorem)
st-Conn can be solved by a deterministic algorithm in $O((\log n)^2)$ space.

Consider the following recursive algorithm for determining whether there is a path from a to b of length at most i.

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path$(a, v, \lfloor i/2 \rfloor)$
2. Path$(v, b, \lceil i/2 \rceil)$

if such an v is found, then accept, else reject.

The maximum depth of recursion is $\log n$, and the number of bits of information kept at each stage is $3 \log n$.

Savitch's Theorem

The space efficient algorithm for st-Conn used on the configuration graph of a nondeterministic machine shows:

$$\mathrm{NSPACE}(f) \subseteq \mathrm{SPACE}(f^2)$$

for $f(n) \geq \log n$.

This yields

$$\mathrm{PSPACE} = \mathrm{NPSPACE} = \text{co-NPSPACE}.$$

Complementation

A still more clever algorithm for st-Conn has been used to show that nondeterministic space classes are closed under complementation:

If $f(n) \geq \log n$, then

$$\text{NSPACE}(f) = \text{co-NSPACE}(f)$$

In particular

$$\text{NL} = \text{co-NL}.$$

Time vs Space

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

and $\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \text{co-NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \text{co-NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \text{co-NPSPACE}$$

It would be easier to prove a more general statement!

Constructible Functions

To prove more general inclusion, we restrict our attention to reasonable time functions.

A complexity class such as $\mathrm{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and
- there is a deterministic machine M which, on any input of length n, replaces the input with the string $0^{f(n)}$, and M runs in time $O(n + f(n))$ and uses $O(f(n))$ *work space*.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

If f and g are constructible functions, then so are $f + g$, $f \cdot g$, 2^f and $f(g)$ (this last, provided that $f(n) > n$).

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every x, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in NTIME(f) is accepted by a machine for which all computations are of length at most $O(f(n))$.

Also, given a Turing machine M and a constructible function f, we can define a machine that simulates M for $f(n)$ steps.

‘

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following, for any constructible f.

- $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$;
- $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$;
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$;
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$;

The first two are straightforward from definitions.

The third is an easy simulation.

The last requires some more work.

Nondeterministic space vs deterministic time

We can use the $O(n^2)$ time algorithm for st-Connectivity to show that:

$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

For any input x of length n, there is a constant c (depending on the number of states and alphabet of M) such that the total number of possible configurations of M within space bounds $f(n)$ is bounded by $n \cdot c^{f(n)}$.

> *Here, $c^{f(n)}$ represents the number of different possible contents of the work space, and n different head positions on the input.*

Space vs Time

Using the $O(n^2)$ algorithm for st-Connectivity, we get that $L(M)$—the language accepted by M—can be decided by a deterministic machine operating in time

$$O(|G|^2) = O((nc^{f(n)})^2) = O(k^{(\log n + f(n))})$$

In particular, this establishes that NL $\subseteq$ P and NPSPACE $\subseteq$ EXP.

Scaling up complexity results

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

A similar argument shows that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{EXP} = \mathsf{NEXP}$.

Summary: A Complexity Zoo

The key players:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

You should also know coNP, coNL, UP, R, RE, BQP (Quantum P)

Bonus contemporary classes: IP, SZK, BPP, FP, FNP, PCP, QMA