Space complexity

Space complexity classes

A language $L \subseteq \{0,1\}^*$ belongs to the class DSPACE(f), if

- there exists a DTM M that accepts L, and a constant c,
- such that $M(x)$ writes to at most $c \cdot f(|x|)$ cells on its work tapes.

The class NSPACE(f) is defined similarly for nondeterministic TMs.

$$\mathsf{PSPACE} = \bigcup_{c \in \mathbb{N}} \mathsf{DSPACE}(\lambda n.n^c) \qquad \mathsf{L} = \mathsf{DSPACE}(\lambda n.\log n)$$

$$\mathsf{NPSPACE} = \bigcup_{c \in \mathbb{N}} \mathsf{NSPACE}(\lambda n.n^c) \qquad \mathsf{NL} = \mathsf{NSPACE}(\lambda n.\log n)$$

Examples

- SAT $\in$ DSPACE$(\lambda n.n)$
- PATH $= \{\langle G, s, t\rangle \mid G$ is dir. graph with path from s to $t\} \in$ NL.

Computing a function and usage of space

- $f : \{0,1\}^* \to \{0,1\}^*$. How to define that it is computable in space g?
- Output is also written on a work tape, how to (dis)count it?

Two possibilities:

- The machine has an extra output tape. It is write-only and the head can only move to the right.
- Languages L_i and L'_i must be in DSPACE(g) for all i, where
  - $L_i = \{x \,|\, |f(x)| \geq i\}$
  - $L'_i = \{x \,|\, x \in L_i \wedge i\text{-th bit of } f(x) \text{ is } 1\}$

Exercise. If $g \in \Omega(\lambda n. \log n)$ then the two variants are the same.

Inclusions and separations

Theorem.
$\mathsf{DTIME}(f) \subseteq \mathsf{DSPACE}(f) \subseteq \mathsf{NSPACE}(f) \subseteq \bigcup_{c\in\mathbb{N}} \mathsf{DTIME}(\lambda n.c^{f(n)})$

Exercise. What is the space complexity of simulating a TM?

Theorem. If $f \in o(g)$ then $\mathsf{DSPACE}(f) \subsetneq \mathsf{DSPACE}(g)$.

Corollary. $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$
We believe all inclusions are strict, but we know only $\mathsf{L} \subsetneq \mathsf{PSPACE}$.

PSPACE-**completeness**

A language L is PSPACE-hard if for any $L' \in \mathrm{PSPACE}$ we have $L' \leq_{\mathrm{m}}^{\mathrm{P}} L$. Language L is PSPACE-complete if it is PSPACE-hard and belongs to PSPACE.

Theorem. The language

$$\{\langle M, x, 1^n \rangle \mid \text{DTM } M \text{ accepts } x \text{ in space } n\}$$

is PSPACE-complete.

Quantified Boolean formulas

A quantified Boolean formula is one of

$$
\begin{aligned}
\phi &= x & FV(\phi) &= \{x\} \\
\phi &= \neg\phi' & FV(\phi) &= FV(\phi') \\
\phi &= \phi_1 \text{ op } \phi_2 & FV(\phi) &= FV(\phi_1) \cup FV(\phi_2) \\
\phi &= Qx\,\phi' & FV(\phi) &= FV(\phi') \backslash \{x\}
\end{aligned}
$$

where Q is either $\forall$ or $\exists$.

Evaluating QBF-s

ϕ defines a Boolean function $[\![\phi]\!]$ from $FV(\phi) \to \mathbb{B}$ to $\mathbb{B}$.

$$
\begin{aligned}
[\![x]\!](V) &= V(x) \\
[\![\neg\phi]\!](V) &= \neg([\![\phi]\!](V)) \\
[\![\phi_1 \,\mathrm{op}\, \phi_2]\!](V) &= [\![\phi_1]\!](V)\ [\![\mathrm{op}]\!]\ [\![\phi_2]\!](V) \\
[\![\forall x\, \phi]\!](V) &= [\![\phi]\!](V[x \mapsto \textsf{true}]) \wedge [\![\phi]\!](V[x \mapsto \textsf{false}]) \\
[\![\exists x\, \phi]\!](V) &= [\![\phi]\!](V[x \mapsto \textsf{true}]) \vee [\![\phi]\!](V[x \mapsto \textsf{false}])
\end{aligned}
$$

Let TQBF $= \{x \text{ is QBF} \mid FV(x) = \emptyset, [\![x]\!]() = \text{true}\}$

All quantifiers can be moved to the front of the formula without increasing the length.

TQBF **is** PSPACE-**complete**

Theorem. TQBF is PSPACE-complete.

Lemma. TQBF $\in$ PSPACE.

Lemma. TQBF is PSPACE-hard.
We reduce $\{\langle M, x, 1^n \rangle \,|\, \text{DTM } M \text{ accepts } x \text{ in space } n\}$ to TQBF

- Recall succinct representations of computation graphs:
  - A configuration encoded in m bits, where $m \leq p(n)$.
  - Formula $S(u_1, \ldots, u_m)$, expressing "being a conf."
  - Formula $R(u_1, \ldots, u_m, u'_1, \ldots, u'_m)$ expressing $C \to C'$.
  - Formulas Φ°, $\Phi^\bullet$ expressing sets of initial and final configurations.
  - Size of everything bounded by $p(n)$.

Reachability

We want to write the formula $R_i(u_1, \ldots, u_m, u'_1, \ldots, u'_m)$ expressing $C \xrightarrow{*} C'$ in at most 2^i steps.

$$R_0(u_1, \ldots, u_m, u'_1, \ldots, u'_m) = \\ R(u_1, \ldots, u_m, u'_1, \ldots, u'_m) \vee (u_1 = u'_1 \wedge \cdots \wedge u_m = u'_m)$$

$$R_i(u_1, \ldots, u_m, u'_1, \ldots, u'_m) = \exists u''_1, \ldots, u''_m : S(u''_1, \ldots, u''_m) \wedge \\ R_{i-1}(u_1, \ldots, u_m, u''_1, \ldots, u''_m) \wedge R_{i-1}(u''_1, \ldots, u''_m, u'_1, \ldots, u'_m)$$

Now $|R_i| = O(2^n) \cdot |R|$. This is bad.

$$R_i(u_1, \ldots, u_m, u'_1, \ldots, u'_m) = \exists u''_1, \ldots, u''_m : S(u''_1, \ldots, u''_m) \wedge \\ \forall v_1, \ldots, v_m, v'_1, \ldots, v'_m : \left(\left(\bigwedge_{k=1}^{m} v_k = u_k \wedge \bigwedge_{k=1}^{m} v'_k = u''_k\right) \vee \\ \left(\bigwedge_{k=1}^{m} v_k = u''_k \wedge \bigwedge_{k=1}^{m} v'_k = u'_k\right)\right) \Rightarrow R_{i-1}(v_1, \ldots, v_m, v'_1, \ldots, v'_m)$$

Encoding TM M

Given M, x, 1^n, we

- Construct m, S, R, Φ°, $\Phi^\bullet$.
- Let n' be such, that M has $\leq 2^{n'}$ configurations of size n.
- Output

$$\exists u_1, \ldots, u_m, u'_1, \ldots, u'_m : \Phi^\circ(u_1, \ldots, u_m) \wedge \\ \Phi^\bullet(u'_1, \ldots, u'_m) \wedge R_{n'}(u_1, \ldots, u_m, u'_1, \ldots, u'_m)$$

TQBF **and** NPSPACE

Theorem. TQBF is NPSPACE-complete.
Exercise. Prove it.
Corollary PSPACE $=$ NPSPACE.

PATH **in deterministic space**

Theorem (Savitch). PATH $\in$ DSPACE$(\lambda n. \log^2 n)$.

Proof. Define the function REACH(u, v, i): There is a path from u to v of length at most 2^i.

REACH$(u, v, i) = \exists w :$ REACH$(u, w, i-1) \wedge$ REACH$(w, v, i-1)$.

Each invocation of REACH needs to store a constant number of vertices. The third argument is initially $\lceil \log n \rceil$.

Corollary. NSPACE$(f) \subseteq$ DSPACE$(\lambda n. f(n)^2)$ for any space-constructible function $f \in \Omega(\lambda n. \log n)$.

PSPACE **and game-playing**

- Imagine a two-player game with perfect information
  - A set of possible states, a starting state, possible ending states with indication who won and lost.
  - For each state: possible legal moves for both players.
  - Both players always know the state the game is in.
- When does the first player have a winning strategy?

$$\exists \text{my move } \forall \text{opp.'s move } \exists \text{my move } \ldots \text{I win!}$$

Similar to TQBF.

log-space reductions

Definition. Language L is log-space reducible to language L' (denote $L \leq_{\mathrm{m}}^{\mathrm{L}} L'$) if

- there exists a function $f : \{0,1\}^* \to \{0,1\}^*$, such that
  - $f(x)$ is computable in space $O(\log|x|)$
  - $x \in L$ iff $f(x) \in L'$.

Theorem. $\leq_{\mathrm{m}}^{\mathrm{L}}$ is transitive. If $L \leq_{\mathrm{m}}^{\mathrm{L}} L'$ and $L' \in \mathsf{L}$ then $L \in \mathsf{L}$.
Exercise. Prove it.

NL-completeness

A language L is NL-hard if for any $L' \in$ NL we have $L' \leq^{\mathrm{L}}_{\mathrm{m}} L$. Language L is NL-complete if it is NL-hard and belongs to NL.

Theorem. PATH is NL-complete.
We already know PATH $\in$ NL. Hence only hardness must be shown.

Log-space reduction to PATH

- Let M be a NTM working in space $O(\log n)$.
- We can compute its adjacency matrix in space $O(\log n)$.
- I.e. given NTM M and two configurations C, C' of size $O(\log n)$, we have to decide in space $O(\log n)$ whether $C \to C'$.

DSPACE(0)

two-way finite automata

Same power as (one-way) finite automata.

Least amount of usable memory

Theorem. If $s \in o(\lambda n.\log\log n)$ then $\mathsf{DSPACE}(s) = \mathsf{DSPACE}(0)$.
Proof on blackboard...

$$L = \{\#\mathsf{bit}(1)\#\mathsf{bit}(2)\#\cdots\mathsf{bit}(n)\# \mid n \in \mathbb{N}\}$$

Exercise. Show that $L \in \mathsf{DSPACE}(\lambda n.\log\log n) \setminus \mathsf{DSPACE}(0)$.

Complexity classes of complementary languages

Let $\mathcal{C}$ be a complexity class. The class co$\mathcal{C}$ is

$$\text{co}\mathcal{C} = \{L^c \mid L \in \mathcal{C}\} \ .$$

- For any f we have DTIME(f) = coDTIME(f) and DSPACE(f) = coDSPACE(f).
- coP = P. coPSPACE = PSPACE. coL = L.
- NP and coNP are thought to be different.
  - P = NP would imply NP = coNP. Opposite implication is not known.

Functions computable by NTMs

An NTM M computes the function $f : \{0,1\}^* \to \{0,1\}^*$, if on input x

- each computation path of M ends by
  - M outputting $f(x)$, or
  - M giving up (outputting "don't know")
- at least one computation path of M ends by outputting $f(x)$.

Number of reachable vertices

Given graph G with n vertices and a vertex s. How many vertices can be reached from s?

Theorem (Immerman-Szelepscényi). This can be computed by a NTM in space $O(\log n)$.

Let $S(k)$ be the set of vertices reachable from s in at most k steps.

The following algorithm can be used to compute $S(n-1)$.

Proof of Immerman-Szelepscényi theorem

$|S(0)| := 1$
for $k := 1$ **to** $n-1$ **do**
    compute $|S(k)|$ from $|S(k-1)|$

Proof of Immerman-Szelepscényi theorem

$|S(0)| := 1$
for $k := 1$ **to** $n-1$ **do**
    $|S(k)| := 0$
    for $u \in V(G)$ **do**
        $b := (u \stackrel{?}{\in} S(k))$
        if b **then** $|S(k)| := |S(k)| + 1$

Proof of Immerman-Szelepscényi theorem

$|S(0)| := 1$
for $k := 1$ **to** $n - 1$ **do**
  $|S(k)| := 0$
  for $u \in V(G)$ **do**
    $m := 0$ $b :=$ false
    for $v \in V(G)$ **do**
      $b' := (v \overset{?}{\in} S(k-1))$ -- nondeterministic procedure
      if b' **then**
        $m := m + 1$
        if $(v, u) \in E(G)$ **then** $b :=$ true
    if $m < |S(k-1)|$ **then give up**
    if b **then** $|S(k)| := |S(k)| + 1$

Proof of Immerman-Szelepscényi theorem

$|S(0)| := 1$
for $k := 1$ **to** $n - 1$ **do**
  $|S(k)| := 0$
  for $u \in V(G)$ **do**
    $m := 0$ $b :=$ false
    for $v \in V(G)$ **do**
      $w_0 := s$ $b' :=$ true
      for $p := 1$ **to** $k - 1$ **do**
        choose $w_p \in V(G)$
        $b' := b' \wedge (w_{p-1}, w_p) \in E(G)$
      if $b' \wedge (w_{k-1} = v)$ **then**
        $m := m + 1$
        if $(v, u) \in E(G)$ **then** $b :=$ true
    if $m < |S(k-1)|$ **then give up**
    if b **then** $|S(k)| := |S(k)| + 1$

NSPACE(f) and coNSPACE(f)

Theorem. If f is a space-computable function, and $f \in \Omega(\lambda n. \log n)$, then NSPACE($f$) = coNSPACE($f$).
Proof sketch.

- Let NTM M working in space f decide $L \subseteq \{0,1\}^*$. We need machine M' that decides L^c.
- The computation graph of a NTM M working in space f has at most $c^{f(n)}$ vertices for some c.
- M' uses previous algorithm to compute $|S(c^{f(n)})|$. This takes $c' \cdot f(n)$ space.
- It checks all configurations of M for being included in $S(c^{f(n)})$.
  - If accepting configuration found, then M' rejects.
  - If algorithm gives up, then M' rejects.
  - If no accepting configuration found, M' accepts.

P-**completeness**

Defined through $\leq_{\mathrm{m}}^{\mathrm{L}}$-reduction.

$$\text{CIRCUITVALUE} = \{\langle C, b_1, \ldots, b_k \rangle \mid \text{Boolean circuit } C \text{ evaluates to true on inputs } b_1, \ldots, b_k\}$$

Theorem. CIRCUITVALUE is P-complete.

Oracle Turing Machines

- An Oracle TM (det. or non-det.) M is a TM with
  - A designated tape — the query tape
  - Three designated states q_{query}, q_{yes}, q_{no}.
- An oracle $\mathcal{O}$ is a subset of $\{0,1\}^*$.
- Whenever M running together with $\mathcal{O}$ (denoted $M^{\mathcal{O}}$) goes into state q_{query},
  - the contents of query tape is interpreted as a bit-string x;
  - M goes to state q_{yes} if $x \in \mathcal{O}$. Otherwise M goes to state q_{no}.
  - This takes a single step.

An oracle $\mathcal{O}$ gives us relativized complexity classes $\mathsf{P}^{\mathcal{O}}$, $\mathsf{NP}^{\mathcal{O}}$, etc.

Limits of diagonalization

- The diagonalization proofs used the facts that
  - there is an efficient mapping between bit-strings and TMs
  - efficient universal TMs exist.
- The proofs did not really consider the internal workings of the TMs M_i from the enumeration of all TMs.
- All these proofs would go through also for oracle TMs.
- Can a similar proof decide $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$.

Theorem. There exist $A, B \subseteq \{0,1\}^*$, such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.

The language EXPCOM

Consider the language

$$\text{EXPCOM} = \{\langle M, x, 1^n \rangle \,|\, \text{DTM } M \text{ accepts } x \text{ in } \le 2^n \text{ steps}\} \ .$$

This is a complete language for exponential-time computation.

A computation in $\text{NP}^{\text{EXPCOM}}$ on input of length n would

- non-deterministically choose a certificate of length $\le p(n)$;
- (make up to $p(n)$ steps), solve up to $p(n)$ problems, each requiring up to $2^{p(n)}$ steps.

A deterministic algorithm would need at most
$2^{p(n)} \cdot p(n) \cdot 2^{p(n)} = 2^{(p(n))^2 \log p(n)}$ steps. Fits in EXPCOM.

Thus $P^{\text{EXPCOM}} = \text{NP}^{\text{EXPCOM}}$.

The oracle B

For any $B \subseteq \{0,1\}^*$ let

$$U_B = \{1^n \mid \exists x : |x| = n \wedge x \in B\} \ .$$

For any B we have $U_B \in \mathsf{NP}^B$. **Exercise.** Why?

We'll now construct a language B, such that $U_B \notin \mathsf{P}^B$.

The oracle B

Let $M_1, M_2, \ldots$ be the enumeration of oracle DTM-s.
We will now define partial functions $\varphi_0, \varphi_1, \ldots$ from $\{0,1\}^*$ to $\{\text{yes}, \text{no}\}$, such that

- φ_0 is always undefined.
- Each φ_i is defined only on a finite subset of $\{0,1\}^*$.
- If $\varphi_i(x)$ is defined, then $\varphi_{i+1}(x) = \varphi_i(x)$.
- For each $x \in \{0,1\}^*$ there exists i, such that $\varphi_i(x)$ is defined.

In the end we define

$$B = \{x \in \{0,1\}^* \mid \exists i : \varphi_i(x) = \text{yes}\} \ .$$

Let t be a superpolynomial function, such that $\forall n : t(n) < 2^n$

Constructing φ_{i+1}

- Set $\varphi_{i+1} = \varphi_i$.
- Let $n = (\max_{\varphi_i(x) \text{ is defined}} |x|) + 1$.
- Run $M_{i+1}^{(\cdot)}(1^n)$ for $t(n)$ steps. If M_i queries for x, then
  - If $\varphi_i(x)$ is defined, then answer $\varphi_i(x)$.
  - If $\varphi_i(x)$ is not defined, then answer no.
    - Set $\varphi_{i+1} = \varphi_{i+1}[x \mapsto \text{no}]$.
- If $M_{i+1}^{(\cdot)}$ stops in $t(n)$ steps, then
  - If $M_{i+1}^{(\cdot)}$ accepts, then set $\varphi_{i+1} = \varphi_{i+1}[\{0,1\}^n \mapsto \text{no}]$
  - If $M_{i+1}^{(\cdot)}$ rejects then pick $x \in \{0,1\}^n$ that $M_{i+1}^{(\cdot)}$ did not query, set $\varphi_{i+1} = \varphi_{i+1}[x \mapsto \text{yes}]$.