Savitch's Theorem

Theorem 23 (Savitch (1970))

$$\text{REACHABILITY} \in \text{SPACE}(\log^2 n).$$

- Let G be a graph with n nodes.
- For $i \geq 0$, let

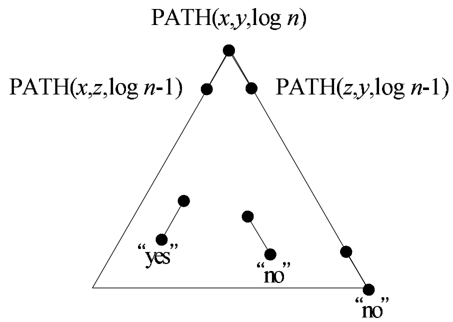
$$\text{PATH}(x, y, i)$$

mean there is a path from node x to node y of length at most 2^i.

- There is a path from x to y if and only if $\text{PATH}(x, y, \lceil \log n \rceil)$ holds.

The Simple Idea for Computing PATH(x, y, i)

- For $i > 0$, $\text{PATH}(x, y, i)$ if and only if there exists a z such that $\text{PATH}(x, z, i-1)$ and $\text{PATH}(z, y, i-1)$.
- For $\text{PATH}(x, y, 0)$, check the input graph or if $x = y$.
- We compute $\text{PATH}(x, y, \lceil \log n \rceil)$ with a depth-first search on a tree with nodes (x, y, i)s.
- Like stacks in recursive calls, we keep only the current path of (x, y, i)s.
- The space requirement is proportional to the depth of the tree, $\lceil \log n \rceil$.



- Depth is $\lceil \log n \rceil$, and each node (x, y, i) needs space $O(\log n)$.
- The total space is $O(\log^2 n)$.

The Algorithm for PATH(x, y, i)

```
if i = 0 then
  if x = y or (x, y) ∈ G then
    return true;
  else
    return false;
  end if
else
  for z = 1, 2, ..., n do
    if PATH(x, z, i − 1) and PATH(z, y, i − 1) then
      return true;
    end if
  end for
  return false;
end if
```

The Relation between Nondeterministic Space and Deterministic Space Only Quadratic

Corollary 24 *Let $f(n) \geq \log n$ be proper. Then*

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

- Apply Savitch's theorem to the configuration graph of the NTM on the input.
- From p. 182, the configuration graph has $O(c^{f(n)})$ nodes; hence each node takes space $O(f(n))$.
- But if we supply the whole graph before applying Savitch's theorem, we get $O(c^{f(n)})$ space!

The Relation between Nondeterministic Space and Deterministic Space Only Quadratic (concluded)

- The way out is not to generate the graph at all.
- Instead, keep the graph implicit.
- We check for connectedness only when $i = 0$, by examining the input string.
- Specifically, given configurations x and y, we go over the Turing machine's program to determine if there is an instruction that can turn x into y in one step.

Implications of Savitch's Theorem

- PSPACE = NPSPACE.
- Nondeterminism is less powerful with respect to space.
- It may be very powerful with respect to time as it is not known if P = NP.

Nondeterministic Space Is Closed under Complement

- Closure under complement is trivially true for deterministic complexity classes (p. 169).
- It is proved in the text that[a]

$$\text{coNSPACE}(f(n)) = \text{NSPACE}(f(n)).$$

- So

$$\begin{aligned} \text{coNL} &= \text{NL}, \\ \text{coPSPACE} &= \text{NPSPACE}. \end{aligned}$$

- But there are still no hints of coNP = NP.

[a] Szelepscényi (1987) and Immerman (1988).

The Immerman-Szelepscényi Theorem

Theorem 25 *Given a graph G and a node x, the number of nodes reachable from x in G can be computed by an NTM within space $O(\log n)$.*

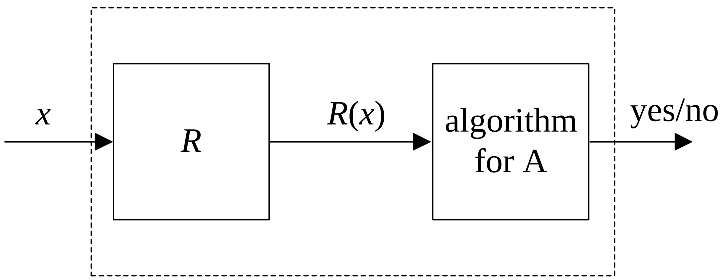
Corollary 26 *If $f(n) \geq \log n$ is proper, then*

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n)).$$

Degrees of Difficulty

- When is a problem more difficult than another?
- B **reduces to** A if there is a transformation R which for every input x of B yields an equivalent input $R(x)$ of A.
  - The answer to x for B is the same as the answer to $R(x)$ for A.
  - There must be restrictions on the complexity of computing R.
  - Otherwise, $R(x)$ might as well solve B.
- Problem A is at least as hard as problem B if B reduces to A.

Reduction



Solving problem B by calling the algorithm for problem *once* and *without* further processing its answer.

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in space $O(\log n)$.
- Furthermore, for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is said to be a (**Karp**) **reduction** from L_1 to L_2.
- Note that by Theorem 22 (p. 179), R runs in polynomial time.

A Paradox?

- Degree of difficulty is not defined in terms of *absolute* complexity.
- A language $B \in \text{TIME}(n^{99})$ may be "easier" than a language $A \in \text{TIME}(n^3)$.
- This happens when B is reducible to A.
- In this case, it is necessary that $|\, R(x) \,| = \Omega(n^{33})$ or that R runs in time $\Omega(n^{99})$ if
$$B \notin \text{TIME}(n^k)$$
for any $k < 99$.

Reduction of HAMILTONIAN PATH to SAT

- Given a graph G, we shall construct a CNF $R(G)$ such that $R(G)$ is satisfiable if and only if G has a Hamiltonian path.
- Suppose G has n nodes: $1, 2, \ldots, n$.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- x_{ij} means "node j is the ith node in the Hamiltonian path."

The Clauses of $R(G)$

1. Each node j must appear in the path.
   - $x_{1j} \vee x_{2j} \vee \cdots \vee x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \vee \neg x_{kj}$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \vee x_{i2} \vee \cdots \vee x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \vee \neg x_{ik}$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \vee \neg x_{k+1,j}$ for all $(i, j) \notin G$ and $k = 1, 2, \ldots, n-1$.

The Proof

- $R(G)$ can be computed efficiently.
- Suppose $T \models R(G)$.
- Clauses of 1 and 2 imply that for each j, there is a unique i such that $T \models x_{ij}$.
- Clauses of 3 and 4 imply that for each i, there is a unique j such that $T \models x_{ij}$.
- So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.
- Clauses of 5 guarantees that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.

The Proof (concluded)

- Conversely, suppose G has a Hamiltonian path
$$(\pi(1), \pi(2), \ldots, \pi(n)),$$
where π is a permutation.
- Clearly, the truth assignment
$$T(x_{ij}) = \texttt{true} \text{ if and only if } \pi(i) = j$$
satisfies all clauses of $R(G)$.

Reduction of REACHABILITY to CIRCUIT VALUE

- Note that both problems are in P.
- Given a graph $G = (V, E)$, we shall construct a *variable-free* circuit $R(G)$.
- The output of $R(G)$ is true if and only if there is a path from node 1 to node n in G.
- Idea: the Floyd-Warshall algorithm.

The Gates

- The gates are
  - g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$.
  - h_{ijk} with $1 \leq i, j, k \leq n$.
- g_{ijk}: There is a path from node i to node j without passing through a node bigger than k.
- h_{ijk}: There is a path from node i to node j passing through k but not any node bigger than k.
- Input gate $g_{ij0} = \texttt{true}$ if and only if $i = j$ or $(i, j) \in E$.

The Construction

- h_{ijk} is an AND gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$, where $k = 1, 2, \ldots, n$.
- g_{ijk} is an OR gate with predecessors $g_{i,j,k-1}$ and $h_{i,j,k}$, where $k = 1, 2, \ldots, n$.
- g_{1nn} is the output gate.
- Interestingly, $R(G)$ uses no $\neg$ gates: It is a **monotone circuit**.
- The depth of $R(G)$ is $O(n)$, which can be improved.

Reduction of CIRCUIT SAT to SAT

- Given a circuit C, we shall construct a boolean expression $R(C)$ such that $R(C)$ is satisfiable if and only if C is satisfiable.
  - $R(C)$ will turn out to be a CNF.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
- Each gate of C will be turned into equivalent clauses of $R(C)$.
- Recall that clauses are $\wedge$ed together.

The Clauses of $R(C)$

g is a variable gate x: Add clauses $(\neg g \lor x)$ and $(g \lor \neg x)$.

- Meaning: $g \Leftrightarrow x$.

g is a `true` gate: Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

g is a `false` gate: Add clause $(\neg g)$.

- Meaning: g must be false to make $R(C)$ true.

g is a $\neg$ gate with predecessor gate h: Add clauses $(\neg g \lor \neg h)$ and $(g \lor h)$.

- Meaning: $g \Leftrightarrow \neg h$.

The Clauses of $R(C)$ (concluded)

g is a $\lor$ gate with predecessor gates h and h': Add clauses $(\neg h \lor g)$, $(\neg h' \lor g)$, and $(h \lor h' \lor \neg g)$.

- Meaning: $g \Leftrightarrow (h \lor h')$.

g is a $\land$ gate with predecessor gates h and h': Add clauses $(\neg g \lor h)$, $(\neg g \lor h')$, and $(\neg h \lor \neg h' \lor g)$.

- Meaning: $g \Leftrightarrow (h \land h')$.

g is the output gate: Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

Composition of Reductions

Proposition 27 *If R_{12} is a reduction from L_1 to L_2 and R_{23} is a reduction from L_2 to L_3, then the composition $R_{12} \cdot R_{23}$ is a reduction from L_1 to L_3.*

- Clearly $x \in L_1$ if and only if $R_{23}(R_{12}(x)) \in L_3$.
- How to compute $R_{12} \cdot R_{23}$ in space $O(\log n)$?
  - Generating $R_{12}(x)$ before feeding it to R_{23} may consume too much space because $R_{12}(x)$ is on a work string.[a]

[a] This would not be a problem if we had required reductions to be in P instead of L.

The Proof (concluded)

- The trick is to let R_{23} drive the computation.
- It asks R_{12} to deliver each bit of $R_{12}(x)$ when needed.
- When R_{23} wants the ith bit, $R_{12}(x)$ will be simulated until the ith bit is available; the beginning $i-1$ bits should not be written to the string.
- This is feasible as $R_{12}(x)$ is produced in a *write-only* manner.
  - The ith output bit of $R_{12}(x)$ is well-defined because once it is written, it will never be overwritten.

Completeness[a]

- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element?
- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if every $L' \in \mathcal{C}$ can be reduced to L.
  - Every complexity class we have seen so far has complete problems!
- Complete problems capture the difficulty of a class because they are the hardest, if they exist.

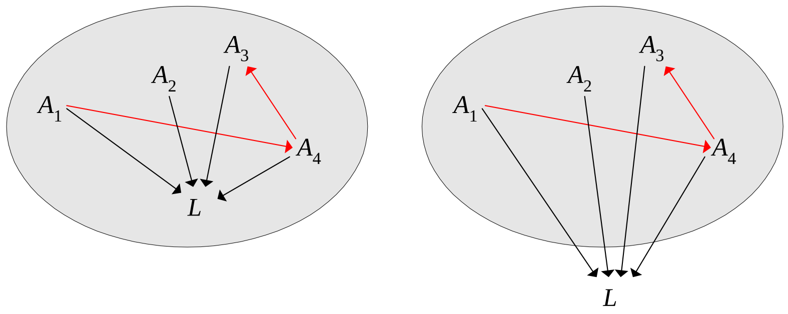
[a]Cook (1971).

Hardness

- Let $\mathcal{C}$ be a complexity class.
- L is **$\mathcal{C}$-hard** if every $L' \in \mathcal{C}$ can be reduced to L.
- It is not required that $L \in \mathcal{C}$.
- If L is $\mathcal{C}$-hard, then by definition, every $\mathcal{C}$-complete problem can be reduced to L.[a]

[a]Thanks to Mr. Ming-Feng Tsai (D92922003).

Illustration of Completeness and Hardness



Closedness under Reduction

- A class $\mathcal{C}$ is **closed under reductions** if whenever L is reducible to L' and $L' \in \mathcal{C}$, then $L \in \mathcal{C}$.
- P, NP, coNP, L, NL, PSPACE, and EXP are all closed under reductions.

Complete Problems and Complexity Classes

Proposition 28 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes such that $\mathcal{C}' \subseteq \mathcal{C}$. Assume $\mathcal{C}'$ is closed under reductions and L is a complete problem for $\mathcal{C}$. Then $\mathcal{C} = \mathcal{C}'$ if $L \in \mathcal{C}'$.*

- Every language $A \in \mathcal{C}$ reduces to $L \in \mathcal{C}'$.
- Because $\mathcal{C}'$ is closed under reductions, $A \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.

Two Immediate Corollaries

Proposition 28 implies that

- P = NP if and only if an NP-complete problem in P.
- L = P if and only if a P-complete problem is in L.

Complete Problems and Complexity Classes

Proposition 29 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes closed under reductions. If L is complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.*

- All languages $\mathcal{L} \in \mathcal{C}$ reduce to $L \in \mathcal{C}'$.
- Since $\mathcal{C}'$ is closed under reductions, $\mathcal{L} \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- The proof for $\mathcal{C}' \subseteq \mathcal{C}$ is symmetric.

Table of Computation

- Let $M = (K, \Sigma, \delta, s)$ be a single-string polynomial-time deterministic TM deciding L.
- Its computation on input x can be thought of as a $|x|^k \times |x|^k$ table, where $|x|^k$ is the time bound.
  - It is a sequence of configurations.
- Rows correspond to time steps 0 to $|x|^k - 1$.
- Columns are positions in the string of M.
- The (i, j)th table entry represents the contents of position j of the string *after* i steps of computation.

Some Conventions To Simplify the Table

- M halts after at most $|x|^k - 2$ steps.
  - The string length hence never exceeds $|x|^k$.
  - Assume a large enough k to make it true for $|x| \geq 2$.
- Pad the table with $\sqcup$s so that each row has length $|x|^k$.
  - The computation will never reach the right end of the table for lack of time.
- If the cursor scans the jth position at time i when M is at state q and the symbol is σ, then the (i, j)th entry is a *new* symbol σ_q.

Some Conventions To Simplify the Table (continued)

- If q is "yes" or "no," simply use "yes" or "no" instead of σ_q.
- Modify M so that the cursor starts not at $\rhd$ but at the first symbol of the input.
- The cursor never visits the leftmost $\rhd$ by telescoping two moves of M each time the cursor is about to move to the leftmost $\rhd$.
- So the first symbol in every row is a $\rhd$ and not a $\rhd_q$.

Some Conventions To Simplify the Table (concluded)

- If M has halted before its time bound of $|x|^k$, so that "yes" or "no" appears at a row before the last, then all subsequent rows will be identical to that row.
- M accepts x if and only if the $(|x|^k - 1, j)$th entry is "yes" for some j.

Comments

- Each row is essentially a configuration.
- If the input $x = 010001$, then the first row is

$$\overbrace{\rhd 0_s 10001 \sqcup \sqcup \cdots \sqcup}^{|x|^k}$$

- A typical row may be

$$\overbrace{\rhd 10100_q 01110100 \sqcup \sqcup \cdots \sqcup}^{|x|^k}$$

- The last rows must look like $\overbrace{\rhd \cdots \text{"yes"} \cdots \sqcup}^{|x|^k}$

A P-Complete Problem

Theorem 30 (Ladner (1975)) CIRCUIT VALUE *is P-complete.*

- It is easy to see that CIRCUIT VALUE $\in$ P.
- For any $L \in$ P, we will construct a reduction R from L to CIRCUIT VALUE.
- Given any input x, $R(x)$ is a variable-free circuit such that $x \in L$ if and only if $R(x)$ evaluates to true.
- Let M decide L in time n^k.
- Let T be the computation table of M on x.

The Proof (continued)

- When $i = 0$, or $j = 0$, or $j = |\,x\,|^k - 1$, then the value of T_{ij} is known.
  - The jth symbol of x or $\sqcup$, a $\triangleright$, and a $\sqcup$, respectively.
  - Three out of four of T's borders are known.

$\triangleright$	a	b	c	d	e	f	$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$

The Proof (continued)

- Consider *other* entries T_{ij}.
- T_{ij} depends on only $T_{i-1,j-1}$, $T_{i-1,j}$, and $T_{i-1,j+1}$.

$T_{i-1,j-1}$	$T_{i-1,j}$	$T_{i-1,j+1}$
	T_{ij}	

- Let Γ denote the set of all symbols that can appear on the table: $\Sigma \cup \{\sigma_q : \sigma \in \Sigma, q \in K\}$.
- Encode each symbol of Γ as an m-bit number, where

$$m = \lceil \log_2 |\,\Gamma\,| \rceil$$

(**state assignment** in circuit design).

The Proof (continued)

- Let binary string $S_{ij1} S_{ij2} \cdots S_{ijm}$ encode T_{ij}.
- We may treat them interchangeably without ambiguity.
- The computation table is now a table of binary entries $S_{ij\ell}$, where

$$0 \le i \le n^k - 1,$$
$$0 \le j \le n^k - 1,$$
$$1 \le \ell \le m.$$

The Proof (continued)

- Each bit $S_{ij\ell}$ depends on only $3m$ other bits:

$T_{i-1,j-1}$:	$S_{i-1,j-1,1}$	$S_{i-1,j-1,2}$	$\cdots$	$S_{i-1,j-1,m}$
$T_{i-1,j}$:	$S_{i-1,j,1}$	$S_{i-1,j,2}$	$\cdots$	$S_{i-1,j,m}$
$T_{i-1,j+1}$:	$S_{i-1,j+1,1}$	$S_{i-1,j+1,2}$	$\cdots$	$S_{i-1,j+1,m}$

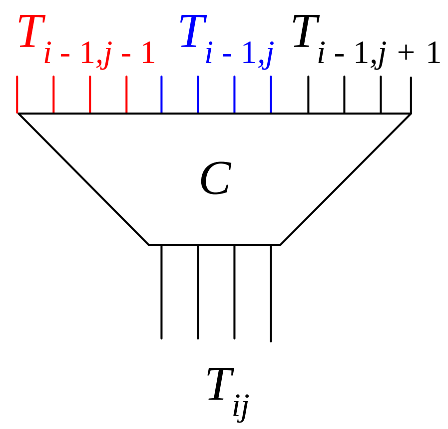
- So there are m boolean functions $F_1, F_2, \ldots, F_m$ with $3m$ inputs each such that for all $i, j > 0$,

$$\begin{aligned} S_{ij\ell} = {} & F_\ell(S_{i-1,j-1,1}, S_{i-1,j-1,2}, \ldots, S_{i-1,j-1,m}, \\ & S_{i-1,j,1}, S_{i-1,j,2}, \ldots, S_{i-1,j,m}, \\ & S_{i-1,j+1,1}, S_{i-1,j+1,2}, \ldots, S_{i-1,j+1,m}). \end{aligned}$$

The Proof (continued)

- These F_i's depend on only M's specification, not on x.
- Their sizes are fixed.
- These boolean functions can be turned into boolean circuits.
- Compose these m circuits in parallel to obtain circuit C with $3m$-bit inputs and m-bit outputs.
  - Schematically, $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}) = T_{ij}$.
  - C is like an ASIC (application-specific IC) chip.

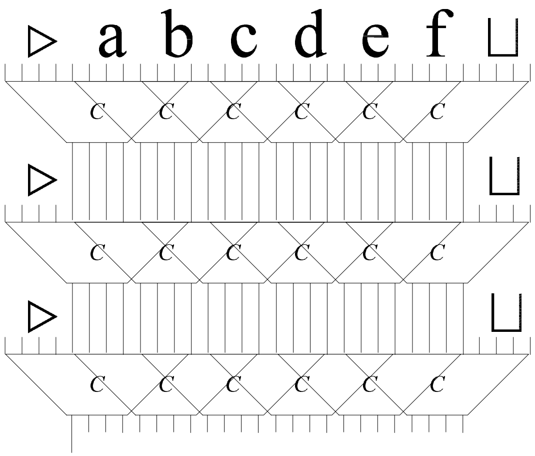
Circuit C



The Proof (concluded)

- A copy of circuit C is placed at each entry of the table.
  - Exceptions are the top row and the two extreme columns.
- $R(x)$ consists of $(|x|^k - 1)(|x|^k - 2)$ copies of circuit C.
- Without loss of generality, assume the output "yes"/"no" (coded as 1/0) appear at position $(|x|^k - 1, 1)$.

The Computation Tableau and $R(x)$



A Corollary

The construction in the above proof shows the following.

Corollary 31 *If* $L \in TIME(T(n))$, *then a circuit with* $O(T^2(n))$ *gates can decide if* $x \in L$ *for* $|x| = n$.