11. P and NP

Goal: Understand the computational problems that can be solved using polynomial time (and potentially non-determinism).

Roughly, P:
- Checking correctness of a proof / a guess.
- Classically considered feasible (Cobham '64).
- About (deterministically) computing (function) values.

Strong form of Church-Turing thesis:
Every physically realizable model* can be simulated by a TM with polynomial overhead
So aliens will have the same notion of P as we do.

* Controversial due to quantum computing.
Realizability not clear.

NP:
- Finding a proof.
- Nowadays considered feasible in practice.
- Puzzles / search problems (that appear to require brute-force).

Milestones:
- Cook '71 & Levin '73: SAT is NP-hard.
- Karp '72: 21 NP-complete problems.
- Ladner '75: CVP is P-complete.

11.1 The Circuit Value Problem (CVP)

Goal:
- Show that CVP is P-complete under logspace reductions.
- Plays the role for $P = NL$ or $P = L$ as the Cook & Levin result for $P = NP$.

Problem:
- Will be the first P-hard problem, so we need reductions from all other problems in P.

Definition:

A Boolean circuit is a program consisting of finitely many assignments

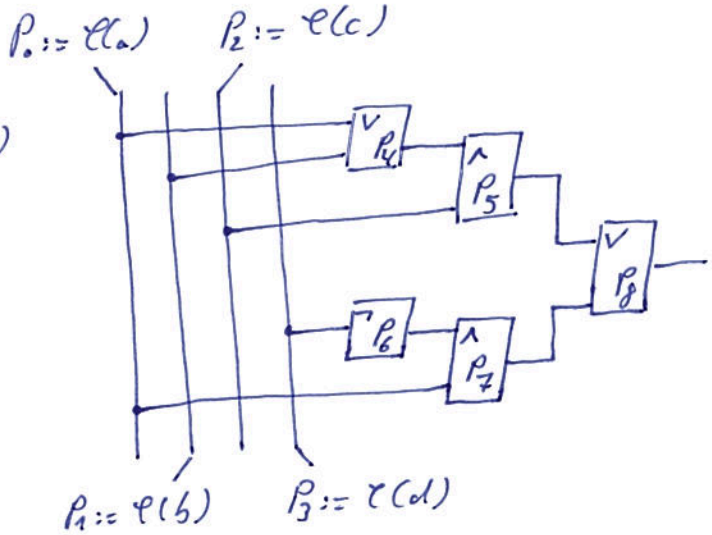
$$P_k := 0/1 \;/\; P_i \vee P_j \;/\; P_i \wedge P_j \;/\; \neg P_i$$

with $i, j < k$ and $i, j, k \in \mathbb{N}$.
Each P_i appears on the left-hand side of exactly one assignment.

Note:
- A circuit can be understood as a DAG where a P_i can be used in multiple later P_k. In this sense, a circuit is more expressive than a Boolean formula.
- A Boolean formula in turn can be represented as a circuit but every P_i (except for the ones assigned 0/1) will be used precisely once. Indeed, the circuit will have the shape of a tree.

Example:

$$\underbrace{\underbrace{(\underbrace{(a \vee b)}_{P_4} \wedge c)}_{P_5} \vee \underbrace{(\underbrace{\neg d}_{P_6} \wedge a)}_{P_7}}_{P_8}$$



Theorem (Ladner '75):

CVP is P-complete under $\leq_m^{\log}$.

Note:
- The analogue problem (evaluation) for Boolean formulas is in L.
- Try to understand why there does not seem to be a logspace reduction of CVP to the corresponding problem for Boolean formulas.

Intuitively, one would like to replace an assignment of the form

$P_k := P_i \wedge P_j$ by $P_k \Leftrightarrow P_i \wedge P_j$.

What is missing?

Proof:

CVP $\in$ P (typically called membership in P):

The circuit can be evaluated in deterministic polynomial time. ✓

CVP is P-hard:

Goal: We have to reduce every problem $\Pi \in P$ to CVP.

Let Π be such a problem and let M be a single-type polynomial-time-bounded DTM that decides Π, say in n^c time.

Wlog we can assume the type alphabet to be some fixed Δ.

Idea: Let x be an input of length n.

- We think about the successive configurations of M on input x as being arranged in an $(n+1) \times (n+1)$ matrix R.
  - ↳ The ith row is a string in $(\Delta \cup (\Delta \times Q))$ describing the configuration at time i.
  - ↳ The jth column describes what is going on in cell j throughout the history of the computation.
- Note that the configurations can be bounded in length by $n^c + 1$, because in every step we can only write one cell.
- To give an example, we would have successive rows

row i $\quad \$ \; a \; b \; a \; a \; b \; \binom{a}{p} \; b \; a \; a \; b \; a \; b \; \sqcup \; \sqcup$, if $\delta(p,a) = (q, b, R)$.

row $i+1$ $\quad \$ \; a \; b \; a \; a \; b \; b \; \binom{b}{q} \; a \; a \; b \; a \; b \; \sqcup \; \sqcup$

Key: · The key observation is that computation is local in the sense that a TM examines and changes only few bits of its tape.

· Technically, we will specify the matrix R in terms of local consistency conditions (over $(n^c+1)\times(n^c+1)$ matrices over $\Delta \cup (\Delta \times Q)$).

Each condition is a relation on the entries of the matrix in a small neighborhood of some location i,j.

The conjunction of these constraints determines R uniquely.

Construction: · Boolean variables

$P_{i,j}^a$:= At time i at cell j we have symbol a.

$Q_{i,j}^p$:= At time i the TM is at cell j and in state p,

$0 \le i,j \le n^c$.

· The tape entries change from configuration $i-1$ to configuration i according to the following constraints ($1 \le i \le n^c$, $0 \le j \le n^c$):

$$P_{i,j}^b := \bigvee_{\delta(p,a)=(q,b,d)} \left(Q_{i-1,j}^p \wedge P_{i-1,j}^a \right) \quad \text{// Write cell } j \text{ at time } i-1$$

$$\vee \left(P_{i-1,j}^b \wedge \bigwedge_{p \in Q} \neg P_{i-1,j}^p \right) \quad \text{// Do not write cell } j \text{ at time } i-1.$$

· The head positions change from configuration $i-1$ to configuration i as follows ($1 \le i \le n^c$, $1 \le j \le n^c$ (ignoring the left-end marker for now)):

$$Q_{i,j}^q := \bigvee_{\delta(p,a)=(q,b,R)} \left(Q_{i-1,j-1}^p \wedge P_{i-1,j-1}^a \right)$$

$$\vee \bigvee_{\delta(p,a)=(q,b,L)} \left(Q_{i-1,j+1}^p \wedge P_{i-1,j+1}^a \right).$$

For the leftmost cell, we only get there from the right.
For the rightmost cell, vice versa.

- It remains to define the first row of the matrix,
i.e., the $P^a_{0,j}$ and $Q^p_{0,j}$ as defined by the initial configuration:

$$P^{\#}_{0,0} := 1$$
$$P^{b}_{0,0} := 0, \quad b \in \Delta \setminus \{\#\}.$$

$$\left.\begin{aligned} P^{a_j}_{0,j} &:= 1, \\ P^{b}_{0,j} &:= 0, \quad b \in \Delta \setminus \{a_j\} \end{aligned}\right\} 1 \le j \le n.$$

$$\left.\begin{aligned} P^{\sqcup}_{0,j} &:= 1, \\ P^{b}_{0,j} &:= 0, \quad b \in \Delta \setminus \{\sqcup\} \end{aligned}\right\} n+1 \le j \le n^c.$$

$$Q^{p_0}_{0,0} := 1, \quad p_0 \text{ initial state.}$$

$$Q^{q}_{0,0} := 0, \quad q \in Q \setminus \{p_0\}.$$

$$Q^{q}_{0,j} := 0, \quad q \in Q, \ 1 \le j \le n^c.$$

Assuming the head moves back to the left
when the computation finishes,
the Boolean value of

$$P := Q^{p_{acc}}_{n^c,0} \qquad (p_{acc} \text{ the accept state})$$

determines whether M accepts input x.

Lemma:
- M accepts x iff CVP returns 1 for P.
- The construction can be done in logspace.
(Although the circuit is polynomial in size, it is highly uniform in the sense that it is built from identical pieces (that only differ in the indices).

Remark:
The result is a good example for the power of Abstraction.
It will also hold for Java and C++ rather than TMs,
but it would have been more difficult
to discover it for these models.

11.2 Cook & Levin's Theorem

Goal: Show that SAT is NP-complete (under $\leq_m^{\log}$).

Problems:
- In SAT, the input is not provided.
  - ↳ Is there a satisfying assignment?
- Circuits are different from Boolean formulas
  - ↳ Share subexpressions, values used more than once.
  - ↳ We discussed this for evaluation above.
  - ⇒ For satisfiability, this difference does not matter.
    Why not? Why this difference with evaluation?

Definition:
A Boolean circuit with unspecified inputs
is a Boolean circuit that may also use

$$P_i := ?,$$

denoting inputs whose values are not specified.

Theorem (Cook '71, Levin '73):
SAT is $\leq_m^{\log}$-complete for NP.

Proof:
- The problem is in NP because we can guess a truth assignment (of polynomial size)
  and verify it in polynomial time (certificate definition of NP).

- To show hardness, let $R \in NP$ and let M be an NTM that decides R and runs in n^c time.
  Wlog. assume M is binary branching.
  This means a path of M is specified by a string from $\{0,1\}^{n^c}$.
- Let M' be the DTM that takes as input $x\#y$,
  where $|y| = |x|^c$.
  Machine M' runs M on input x
  and uses y to resolve non-deterministic choices.
  The new machine will accept if M accepts
  (on the path specified by y).
- We apply the construction from Ladner's theorem
  and obtain a CVP instance where
  the circuit has value 1 iff M' accepts $x\#y$.
  Note from the construction that if
  $|z| = |y| = n^c$,
  the circuit constructed for $x\#z$ will coincide with $x\#y$,
  except for the inputs corresponding to y.
  We make these inputs unspecified and obtain a circuit

  $$C(P_1 := ?, \dots, P_{n^c} := ?).$$

  Now:
  M accepts x iff $\exists y_1, \dots, y_n : C(y_1, \dots, y_n) = 1$.
- It remains to transform the circuit into a Boolean formula.
  We replace every assignment
  $P_i := E$ by $P_i \Leftrightarrow E$
  and take the conjunction of the resulting clauses.

The formula is satisfiable iff M accepts x. □

Remark: SAT remains NP-hard for CNFs with 3 literals per clause, 3SAT.

On the Reduction:

Levin:
- The reduction f from an NP-problem A to SAT not only satisfies
  $x \in A$ iff $f(x) \in SAT$,
  but the proof yields an efficient way of transforming a certificate for x to a satisfying assignment for $f(x)$, and vice versa.

A reduction with this property is called a Levin reduction.

Parsimonious:
- One can also modify the proof so that there is a bijection between
  - the certificates for x and
  - the certificates for $f(x)$.
- A reduction that guarantees that the number of certificates for x is equal to the number of certificates for $f(x)$ is called parsimonious.
- Parsimonious reductions are important in the study of counting complexity — the complexity of counting the number of certificates for an instance of an NP problem.

Most NP-complete problems have parsimonious Levin reductions from all NP problems.

11.3 Context-free Languages, Dynamic Programming, and P

Goal: · Show that every context-free language is in P.
· Introduce the algorithmic technique of dynamic programming.

Dynamic programming: · Accumulate information about smaller subproblems to solve larger problems.
· Store solution to subproblems to avoid recomputing them (make a table where key we stored). (memoization)

Examples: · Fibonacci:

Naive: $fib(5) = fib(4) + fib(3)$
$= (fib(3) + fib(2)) + (fib(2) + fib(1))$
$= ((fib(2) + fib(1)) + fib(2)) + (fib(2) + fib(1))$
$= \dots$

Dynamic programming: Store $mem(0) := 0$, $mem(1) := 1$
and set $mem(n) := mem(n-1) + mem(n-2)$.

· Shortest path.

Idea: Memoization, and dynamic programming in general, is like computing a fixed point on auxiliary information.

Definition:
The problem $L(G)$ with G a context-free grammar in Chomsky normal form ($A \to BC$ or $A \to a$) is the membership problem for the language:

Given: Input string w.
Question: Does $w \in L(G)$ hold?

Idea for dynamic programming:
- The subproblems determine for each non-terminal A in G and for every infix v of w whether $A \Rightarrow^* v$.

Table:
- The algorithm enters the solution in an $n \times n$ table, $n = |w|$.

For $i \leq j$, we have

table(i,j) := Non-terminals that generate $w_i \dots w_j$.

For $i > j$, the table entries are unused.

Filling:
- Fill the table entries for all substrings of w.
- Increase in length:
  - Start from substrings of length 1.
  - Continue with substrings of length 2.
  - ...
- Key: Use entries for the shorter lengths to determine the entries for the longer lengths.

Accept: If startsymbol S is in the set table$(1,n)$.

Details on filling:
- Assume we have already determined which non-terminals generate all substrings of length $\leq k$.
- To determine whether A generates v of length $k+1$,

$v = v_1 \dots v_{k+1}$,

split v into two non-empty pieces.
There are k possible ways of splitting v.
- For each split position m, $v_1 \dots v_m$, $v_{m+1} \dots v_{k+1}$,
examine all rules

$A \to BC$.

Check whether • B generates $v_n \dots v_m$ and
• C generates $v_{m+1} \dots v_{k-1}$.
If so, add A to the entry for v.

Analysis idea:
- There are three nested loops:
  - ↳ Length of substring.
    - ↳ Start index.
      - ↳ Split position.

  Hence, the runtime is $O(n^3)$.
- This needs more care!
  Write down the algorithm, inspect it in detail.

Theorem: Every context-free language is in P.