Diagonalization

Our goal: separate interesting complexity classes
Question: could we somehow use diagonalization to resolve P vs. NP?
Only general technique we have (e.g., for hierarchy theorems)
Diagonalization relies on following properties of Turing machines:
- A Turing machine can be represented by a string
- A Turing machine can be simulated by another Turing machine without much overhead in time or space.
- Treats machines as blackboxes: internal workings don’t matter.

Could we use diagonalization to resolve P vs. NP?

Here’s some evidence that this won’t work:
An Oracle Turing machine MA is a modified TM with the ability to query an oracle for language A.
Has special tape called an oracle tape.
When MA writes a string on oracle tape, it is informed whether that string is in A in one step.
Observations:
- NP \subseteq P^{\text{sat}}
- coNP \subseteq P^{\text{sat}} (because deterministic complexity classes are closed under complementation)
- \text{N}^{\text{P}^{\text{sat}}} contains languages we believe are not in NP.
- Example: \{ \phi | \phi \text{ is not a minimal boolean formula} \}

Relativization

An argument “relativized” if it goes through when you give the machine oracle access.
Essentially, diagonalization is a simulation of one TM by another.
Simulation ensures that simulating machine determines behavior of other machine and then behaves differently.
What if you add an oracle?
Simulation proceeds as before => if we could prove P = NP, we could also prove P = \text{NP}^A

Diagonalization and Relativization

Theorem:
There is an oracle B whereby \text{P}^B = \text{NP}^B
There is an oracle A whereby \text{P}^A = \text{NP}^A

SPACE: The next frontier

Quite different from time: space can be reused.
Space complexity of Turing machine M = space used.
+ maximum number of tape cells that M scans as function of input length.
For non-deterministic TM, wherein all branches halt, space complexity defined as maximum number of tape cells scanned on any branch of computation as function of input length.
As usual, use asymptotic notation.
SPACE(f(n)) = \{ L | L is language decided by an O(f(n)) space deterministic TM \}
NSPACE(f(n)) = \{ L | L is language decided by an O(f(n)) space non-deterministic TM \}

Savitch’s Theorem

Savitch’s Thm: Any nondeterministic TM that uses f(n) space can be converted to deterministic TM that uses O(f^2(n)) space.
Idea: Solve yieldability problem.
Given two configurations of the NTM C and C’, together with number t, determine if NTM can get from C to C’ in t steps.
Solve CanYield(C_{start}, C_{accept}, 2O(f(n)) )
Use recursive algorithm, by searching for intermediate configuration.
Savitch’s theorem

**boolean CanYield(c1, c2, t) {**
  if (t \leq 1) return correct answer
  foreach configuration c’ {
    boolean x = CanYield(c1, c’, t/2)
    boolean y = CanYield(c2, c’, t/2)
    if (x and y) return true
  }
  return false
**}

On input w:
- Output the result of CanYield(\text{cstart, caccept}, 2^{O(f(n))})
- Whenever CanYield invokes itself recursively, stores the current stage number and values of c1, c2 and t on stack.
- Every level of recursion uses \(O(f(n))\) space.
- Depth of recursion \(O(f(n))\) => total space \(O(f(n)^2)\)

PSPACE

**P.** Decision problems solvable in polynomial time.

**PSPACE.** Decision problems solvable in polynomial space.

**EXPTIME.** Decision problems solvable in exponential time.

Relationships: \(P \subseteq NP \subseteq PSPACE = NPSPACE\)

Quantified Satisfiability

**QSAT.** Let \(\Phi(x_1, \ldots, x_n)\) be a Boolean CNF formula. Is the following propositional formula true?

\[ \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_n \Phi(x_1, \ldots, x_n) \]

This is called the quantified subproblem.

**Intuition.** Amy picks truth value for \(x_1\), then Bob for \(x_2\), then Amy for \(x_3\), and so on. Can Amy satisfy \(\Phi\) no matter what Bob does?

**Ex.** \((x_1 \vee x_2) \land (x_3 \vee \lnot x_4) \land (\lnot x_5 \vee x_6)\)
- Yes. Amy sets \(x_1\) true; Bob sets \(x_2\) true; Amy sets \(x_3\) to be same as \(x_2\),
- Ex. \((x_1 \vee x_2) \land (\lnot x_3 \vee x_4) \land (\lnot x_5 \vee x_6)\)
- No. If Amy sets \(x_1\) false; Bob sets \(x_2\) false; Amy loses.
  - If Amy sets \(x_1\) true; Bob sets \(x_2\) true; Amy loses.

**Theorem.** QSAT \(\subseteq\) PSPACE.
- Recursively try all possibilities.
- Only need one bit of information from each subproblem.
- Amount of space is proportional to depth of function call stack.

QSAT is in PSPACE

**Theorem.** QSAT \(\subseteq\) PSPACE.
- Recursively try all possibilities.
  - Only need one bit of information from each subproblem.
  - Amount of space is proportional to depth of function call stack.
Planning

Planning Problem

Conditions. Set C = {C1, …, Cn}.
Initial configuration. Subset c0 ⊆ C of conditions initially satisfied.
Goal configuration. Subset c* ⊆ C of conditions we seek to satisfy.
Operators. Set O = {O1, …, Ok}.
To invoke operator Oi, must satisfy certain prereq conditions.
After invoking Oi, certain conditions become true, and certain conditions become false.

PLANNING. Is it possible to apply sequence of operators to get from initial configuration to goal configuration?

Examples.
Many puzzles such as 15-puzzle, Rubik’s cube.
Logistical operations to move people, equipment, materials and robots (software or hardware).

Planning Problem: Binary Counter

Planning example. Can we increment an n-bit counter from the all-zeros state to the all-ones state?

Conditions. C1, …, Cn.
Initial state. c0 = {C1, …, Cn}.
Goal state. c* = {C1, …, Cn}.
Operators. O1, …, On.
To invoke operator Oi, must satisfy C1, …, Ci-1.
After invoking Oi, condition Ci becomes true.
After invoking Oi, conditions C1, …, Ci-1 become false.
Solution. {C1}, {C2}, {C1, C2}, {C3}, {C3, C1}, …
Observation. Any solution requires at least 2n - 1 steps.

Planning Problem: In Exponential Time

Configuration graph G.
- Include node for each of 2^n possible configurations.
- Include edge from configuration c' to configuration c'' if one of the operators can convert from c' to c''.

PLANNING. Is there a path from c0 to c* in configuration graph?

Claim. PLANNING is in EXPTIME.
Pf. Run BFS to find path from c0 to c* in configuration graph.

Note. Configuration graph can have 2^n nodes, and shortest path can be of length 2^n - 1.

Planning Problem: In Polynomial Space

Theorem. PLANNING is in PSPACE.
Pf. Same idea as proof of Savitch’s theorem.
Suppose there is a path from c1 to c2 of length L.
Path from c1 to midpoint and from c2 to midpoint are each ≤ L/2.
Enumerate all possible midpoints.
Apply recursively. Depth of recursion = log2 L.

boolean hasPath(c1, c2, L) {
  if (L ≤ 1) return correct answer
  foreach configuration c' {
    boolean x = hasPath(c1, c', L/2)
    boolean y = hasPath(c2, c', L/2)
    if (x and y) return true
  }
  return false
}

PSPACE-Completeness
**PSPACE-Complete**

PSPACE. Decision problems solvable in polynomial space.

PSPACE-Complete. Problem Y is PSPACE-complete if (i) Y is in PSPACE and (ii) for every problem X in PSPACE, X \( \leq_p Y \).

**Why polynomial reducibility?**

Think about what it means for a problem to be complete for a complexity class:
- one of the hardest problems in the class
- every other problem in the class "easily" reduced to it.
- so reduction must be easy, relative to complexity of typical problems in class.


Theorem. PSPACE \( \subseteq \) EXPTIME.

Pf. Previous algorithm solves QSAT in exponential time, and QSAT is PSPACE-complete.

Summary. P \( \subset \) NP \( \subset \) PSPACE \( \subset \) EXPTIME.

It is known that P \( \subseteq \) EXPTIME, but unknown which inclusion is strict; conjectured that all are.

**PSPACE-Complete Problems**

More PSPACE-complete problems:
- Competitive facility location.
- Natural generalizations of games:
  - Othello, Hex, Geography, Rush-Hour, Instant Insanity
  - Shanghai, go-moku, Sokoban
- Various motion planning and search problems.

---

**Competitive Facility Location**

**Input.** Graph with positive edge weights, and target B.

**Game.** Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors has been selected.

**Competitive facility location.** Can second player guarantee at least B units of profit?

Yes if B = 20; no if B = 25.

---

**Construction.** Given instance \((x_1, \ldots, x_n) = C_1 \land \ldots \land C_k\) of QSAT.

- Include a node for each literal and its negation and connect them.
- at most one of \(x_i\) and its negation can be chosen.
- Choose \(c \ll 2^n\) and put weight \(c\) on literal \(x_i\) and its negation;
- set \(B = c^{n^2} + c^{n^2} + \cdots + c^2 + c^2\).
- ensure variables are selected in order \(x_n, x_{n-1}, \ldots, x_1\).
- At \(k\) is, player 2 will lose by 1 unit: \(c^{n^2} + c^{n^2} + \cdots + c^2 + c^2\).

---

**Claim.** COMPETITIVE-FACILITY is PSPACE-complete.

Pf.

- In PSPACE

To show that it’s complete, we show that QSAT polynomial reduces to it. Given an instance of QSAT, we construct an instance of COMPETITIVE-FACILITY such that player 2 can force a win iff QSAT formula is true.
Give player 2 one last move on which she can try to win. For each clause \( C_j \), add a node with value 1 and an edge to each of its literals. Player 2 can make a last move iff no truth assignment defined alternately by the players satisfies some clause.

Other important results related to space complexity:

- Sipser, Sections 8.4--8.6; Arora, Barak, Section 3.4
  - The classes \( L \) (logspace) and \( NL \) (nondeterministic logspace).
  - \( NL \) completeness:
    - The clique problem and \( 
    \text{3-SAT} \text{-instances} \) are \( NL \)-complete.
  - \( NL \) is closed under complementation.
  - \( NL \) strictly contains \( \text{coNL} \).
  - \( NL \) is contained in \( \text{PSPACE} \).
  - \( \text{SPACE}(n^{c}) \) is contained in \( \text{E}-\text{SPACE} \).
  - \( \text{SPACE}(n^{c}) \) is contained in \( \text{E}^{\text{SPACE}} \).