Complexity Class P

• Deterministic in nature
• Solved by conventional computers in polynomial time
  – $O(1)$ Constant
  – $O(\log n)$ Sub-linear
  – $O(n)$ Linear
  – $O(n \log n)$ Nearly Linear
  – $O(n^2)$ Quadratic
• Polynomial upper and lower bounds
Decision and Optimization Problems

• Decision Problem: computational problem with intended output of “yes” or “no”, 1 or 0
• Optimization Problem: computational problem where we try to maximize or minimize some value
• Introduce parameter k and ask if the optimal value for the problem is at most or at least k. Turn optimization into decision
Complexity Class NP

• Non-deterministic part as well
• choose(b): choose a bit in a non-deterministic way and assign to b
• If someone tells us the solution to a problem, we can verify it in polynomial time
• Two Properties: non-deterministic method to generate possible solutions, deterministic method to verify in polynomial time that the solution is correct.
Circuit-SAT

- Take a Boolean circuit with a single output node and ask whether there is an assignment of values to the circuit’s inputs so that the output is “1”
Knapsack

• Given \( s \) and \( w \) can we translate a subset of rectangles to have their bottom edges on \( L \) so that the total area of the rectangles touching \( L \) is at least \( w \)?
PTAS

- Polynomial-Time Approximation Schemes
- Much faster, but not guaranteed to find the best solution
- Come as close to the optimum value as possible in a reasonable amount of time
- Take advantage of rescalability property of some hard problems
Backtracking

• Effective for decision problems
• Systematically traverse through possible paths to locate solutions or dead ends
• At the end of the path, algorithm is left with \( (x, y) \) pair. \( x \) is remaining subproblem, \( y \) is set of choices made to get to \( x \)
• Initially \( (x, \emptyset) \) passed to algorithm
Algorithm Backtrack(x):

Input: A problem instance x for a hard problem  
Output: A solution for x or “no solution” if none exists

F ← {(x, Ø)}.

while F ≠ Ø do

select from F the most “promising” configuration (x, y)
expand (x, y) by making a small set of additional choices
let (x₁, y₁), ..., (xₖ, yₖ) be the set of new configurations.

for each new configuration (xᵢ, yᵢ) do

perform a simple consistency check on (xᵢ, yᵢ)
if the check returns “solution found” then

return the solution derived from (xᵢ, yᵢ)

if the check returns “dead end” then

discard the configuration (xᵢ, yᵢ)

else

F ← F ∪ {(xᵢ, yᵢ)}.

return “no solution”
Branch-and-Bound

• Effective for optimization problems
• Extended Backtracking Algorithm
• Instead of stopping once a single solution is found, continue searching until the best solution is found
• Has a scoring mechanism to choose most promising configuration in each iteration
Algorithm Branch-and-Bound(x):

**Input:** A problem instance $x$ for a hard optimization problem

**Output:** A solution for $x$ or "no solution" if none exists

1. $F \leftarrow \{(x, \emptyset)\}$.
2. $b \leftarrow \{(+\infty, \emptyset)\}$.

while $F \neq \emptyset$ do

    select from $F$ the most "promising" configuration $(x, y)$

    expand $(x, y)$, yielding new configurations $(x_1, y_1), ..., (x_k, y_k)$

    for each new configuration $(x_i, y_i)$ do

        perform a simple consistency check on $(x_i, y_i)$

        if the check returns "solution found" then

            if the cost $c$ of the solution for $(x_i, y_i)$ beats $b$ then

                $b \leftarrow (c, (x_i, y_i))$

            else

                discard the configuration $(x_i, y_i)$

        if the check returns "dead end" then

            discard the configuration $(x_i, y_i)$

        else

            if $lb(x_i, y_i)$ is less than the cost of $b$ then

                $F \leftarrow F \cup \{(x_i, y_i)\}$.

            else

                discard the configuration $(x_i, y_i)$

    return $b$
Polynomial-Time Reducibility

• Language $L$ is polynomial-time reducible to language $M$ if there is a function computable in polynomial time that takes an input $x$ of $L$ and transforms it to an input $f(x)$ of $M$, such that $x$ is a member of $L$ if and only if $f(x)$ is a member of $M$.

• Shorthand, $L^{\text{poly}}M$ means $L$ is polynomial-time reducible to $M$. 

NP-Hard and NP-Complete

• Language M is NP-hard if every other language L in NP is polynomial-time reducible to M
• For every L that is a member of NP, $L^{\text{polyM}}$
• If language M is NP-hard and also in the class of NP itself, then M is NP-complete