CS3230

ASYMPTOTIC ANALYSIS

Word-RAM model

- Word is a collection of few bytes
- Word is the basic storage unit of RAM, which can be viewed as huge array of words
- Each input item is stored in binary format
- An arbitrary location of RAM can be accessed in the same time irrespective of the location
- Data and program fully reside in RAM
- Each arithmetic or logical operation involving a **constant number** of words takes **constant number of cycles (steps)** by the CPU

Big-O notation

Upper bound ($\geq$)
We write $f(n) = O(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq \mathbf{f(n)} \leq cg(n)$$
for all $n \geq n_0$.

Lower bound ($\leq$)
We write $f(n) = \Omega(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq cg(n) \leq \mathbf{f(n)}$$
for all $n \geq n_0$.

Tight bound We write $f(n) = \Theta(g(n))$ if for some positive constants c_1, c_2, n_0,
$$0 \leq c_1 g(n) \leq \mathbf{f(n)} \leq c_2 g(n)$$
for all $n \geq n_0$.

Strict upper bound ($>$)
We write $f(n) = o(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq \mathbf{f(n)} < cg(n)$$
for all $n \geq n_0$.

Strict lower bound ($<$)
We write $f(n) = \omega(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq cg(n) < \mathbf{f(n)}$$
for all $n \geq n_0$.

Properties

Set notations

- The notations are really just sets, e.g.
$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq c(g(n)) \quad \forall n \geq n_0\}$$
- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Transitivity

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

Reflexivity

$$f(n) = \Theta(f(n)) \quad f(n) = O(f(n)) \quad f(n) = \Omega(f(n))$$

Symmetry

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$

Complementarity

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

Useful facts

- Degree-k polynomials are $O(n^k)$, $o(n^{k+1})$, and $\omega(n^{k-1})$
- Polys dominate logs: $(\log n)^{100} = o(n^{.0001})$
- Exponentials dominate polys: $n^{1000} = o(2^{.001n})$
- $\max(f(n), g(n)) = \Theta(f(n) + g(n))$

Exponentials

- For constants $k > 0, a > 1, n^k = o(a^n)$
- Exponentials of different bases differ by an **exponential factor**
- $2^{n+5} = O(2^n)$, but $2^{5n} \neq O(2^n)$

Properties

$$a^{-1} = \frac{1}{a} \qquad a^m a^n = a^{m+n}$$
$$(a^m)^n = a^{mn} \qquad e^x \geq 1 + x$$

Logarithms

- Binary log: $\lg n = \log_2 n$
- Natural log: $\ln n = \log_e n$
- Exponentiation: $\lg^k n = (\lg n)^k$
- Composition: $\lg \lg n = \lg(\lg n)$
- Base of log does not matter in asymptotics: $\lg n = \Theta(\ln n) = \Theta(\log_{10} n)$

Properties

$$a = b^{\log_b a} \qquad \log_b \frac{1}{a} = -\log_b a$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a \qquad \log_b a = \frac{1}{\log_a b}$$
$$\log_b a = \frac{\log_c a}{\log_c b} \qquad a^{\log_b c} = c^{\log_b a}$$

Overview

$$1 < \log n < \sqrt{n} < n < n \log n < n^2$$
$$< n^3 < 2^n < 2^{2n} < n! < n^n$$

Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$
$$\log(n!) = \Theta(n \lg n)$$

Arithmetic series

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$$

Geometric series

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \text{ when } |x| < 1$$

Harmonic series

$$\sum_{k=1}^{\infty} \frac{1}{k} = \ln n + O(1)$$

Misc

$$\lg(\lg n)! = \Theta(\lg n \lg \lg n)$$
by subbing $\lg n$ into Stirling's approx.
$$\sum_{i=1}^{n-2} \lg \lg(n-i) = \Theta(n \lg \lg n)$$
$$\sum_{i=1}^{\lg n - 1} \lg \lg \frac{n}{2^i} = \Theta(\lg n \lg \lg n)$$
$$n! > \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

- For $T(n) = 2T(\sqrt{n}) + a$, the recursion tree has height $\lg \lg n$. Visualize by applying lg to each element of the recursion tree
- To compare two functions, consider taking the lg of each and compare that instead. This works since lg is strictly increasing

Limits

Assume $f(n), g(n) > 0$.
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = O(g(n))$$
$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = \Theta(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) = \Omega(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \omega(g(n))$$

Epsilon-delta definition Let $f(x)$ be a function defined on an open interval around x_0, where $f(x_0)$ need not be defined. Then
$$\lim_{x \to x_0} f(x) = L$$
if for every ε there exists $\delta > 0$ such that for all x,
$$0 < |x - x_0| < \delta \implies |f(x) - L| < \varepsilon$$

L'Hopital If $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = 0$ or $\pm\infty$,
$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$$

Power of e
$$\lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^{bn+c} = e^{ab}$$

AMORTIZED ANALYSIS

Guarantees the average performance of each op in the worst case

Aggregate method

- Count total cost and divide by number of ops

Queues

- n INSERT and EMPTY operations
- Notice EMPTY is a sequence of DELETES, and DELETES $\leq$ INSERTS
- If there are k INSERTs, then sum of cost of all EMPTYs is $\leq k$
- Total cost $\leq k + k = 2k \leq 2n$ since $k \leq n$. Amortized cost is $O(1)$

Accounting method

- Charge ith operation a fictitious amoritzed cost $c(i)$, that satisfies
$$\sum_{i=1}^{n} t(i) \leq \sum_{i=1}^{n} c(i) \quad \forall n$$
where $t(i)$ is the true cost of the ith operation
- Usually $c(i) > t(i)$, with the extra amount paid stored as credit for future, rare, expensive operations
- Analysis should ensure that there's always enough credit to pay for ture cost
- Always **identify the expensive operation**, which you try to do "free of cost" using stored credit

Queues

- For INSERT, set amortized cost to 2 (true cost is 1)
- For EMPTY, set amortized cost to 0 (true cost is size of queue)
- Whenever an element is inserted, we pay an extra 1. This extra 1 can be used as credit to pay for later deletions
- Total cost is at most 2 $\times$ number of INSERTS $\leq 2n$

Binary increment

- Charge 2 for each $0 \to 1$; Charge 0 for each $1 \to 0$
- Starting from 0, actual cost for n increments is $O(n)$

Potential method

Motivation

- Accounting method tries to eyepower the required amortized cost
- Potential method tries to find some metric that decreases a lot on expensive operations
- Similar idea

Idea

- Define potential function ϕ, where $\phi(i)$ denotes potential at the end of the ith operation
- Must have $\phi(i) \geq 0$ for all i
- Amortized cost of ith op, $c(i)$ is defined as $c(i) = t(i) + \Delta\phi$ where $\Delta\phi = \phi(i) - \phi(i-1)$
- Amortized cost of nth operations
$$\sum_i c(i) = \sum_i t(i) + (\phi(n) - \phi(0)) \geq \sum_i t(i) - \phi(0)$$
- Select suitable ϕ so that for the **costly** operation, $\Delta\phi_i$ is **negative to an extent** that it nullifies or reduces the effect of the actual cost

Binary increment

Working

- Use $\phi(i)$ = number of 1s in the counter after ith increment
- Let L_i be the length of the longest suffix with all 1s
- True cost of ith increment is $1 + L_i$
- $\Delta\phi_i = -L_i + 1$
- Sum of actual cost and $\Delta\phi_i$ is 2, so amortized cost of ith increment is 2

Results

- Starting from 0, actual cost for n increments is $O(n)$
- Starting from t ones, actual cost for n increments is $O(n + t)$

DP

SRTBOT

1. Subproblem definition
2. Relate subproblem solutions recursively
3. Topological order of subproblems to guarantee acyclic
4. Base cases of relation
5. Original problem: solve via subproblems
6. Time analysis

Examples shown

Fibonacci, LCS, Knapsack, Coin change

Terminology

Optimal substructure

An optimal solution to a problem contains optimal solutions to subprobems

Overlapping subproblems

A recursive solution contains a "small" number of distinct subproblems repeated many times

Cut-and-paste proof

Often used to show optimal substructure

1. Suppose your optimal solution is made using suboptimal solutions to subproblems
2. Show that if you were to replace the suboptimal subproblem solutions with optimal subproblem solutions, you would improve your optimal solution
3. Hence assumption is false, and the optimal solution is indeed made using optimal subproblem solutions

Optimal substructure for coin change

Let $M[j]$ denote the minimum number of coins required to change j cents. Let the coins be $d_1, d_2, \cdots, d_k$.

Suppose $M[j] = t$ is optimal, i.e.

$$j = d_{i_1} + d_{i_2} + \cdots + d_{i_t}$$

for some $i_1, \cdots, i_t \in \{1, \cdots, k\}$.

- Consider subproblem j', where $j' = d_{i_1} + d_{i_2} + \cdots d_{i_{t-1}}$, and $M[j'] = t - 1$
- If this were suboptimal, then $M[j'] < t - 1$.
- By cut-and-paste argument, since we just need to add coin d_{i_t} to subproblem j' to reach subproblem j, then
$$M[j] = M[j'] + 1 < t - 1 + 1 = t$$
- Contradicts the claim that $M[j] = t$ is optimal
- Hence the optimal solution is indeed made using optimal solutions to subproblems

GREEDY

Paradigm

1. Recast problem so that only one subproblem needs to be solved at each step
2. Prove greedy-choice property
3. Use optimal substructure to show that we can combine an optimal solution to the subproblem with the greedy choice, to get an optimal solution to the original problem

Examples used

Fractional knapsack, Prim (MST), Choose top k items

Terminology

Refer to above for

- Optimal substructure
- Overlapping subproblems
- Cut-and-paste proof

Greedy-choice property

- Locally optimal choice is also globally optimal
- Many optimal solutions may exist, but there is an optimal solution that made the greedy choice

Fractional knapsack

Optimal substructure

Let x_i be the chosen weight for item i.

Suppose $(x_1, \cdots, x_i \cdots, x_n)$ is an optimal solution to $\big((w_1, v_1), \cdots, (w_i, v_i), \cdots, (w_n, v_n), W\big)$.

WTS: $(x_1, \cdots, x_i - w \cdots, x_n)$ is an optimal soln to $\big((w_1, v_1), \cdots, (w_i - w, v_i), \cdots, (w_n, v_n), W\big)$.

- Suppose better solution for subproblem exists, i.e.
  - some $(y_1, \cdots, y_i, \cdots, y_n)$ weighs $\leq (W - w)$, with better value than $(x_1, \cdots, x_i - w, \cdots, x_n)$
- Then $(y_1, \cdots, y_i + w, \cdots, y_n)$ weighs $\leq W$, and has better value than $(x_1, \cdots, x_i, \cdots, x_n)$
- Contradicts the claim that we had an optimal solution
- Hence the optimal solution is indeed made using optimal solutions to subproblems

Greedy choice property

Let j^* be the item with the maximum value per kg, v_j / w_j.

WTS: There exists an optimal knapsack containing $\min(w_{j^*}, W)$ kgs of item j^*, i.e. as much as possible of item j^*.

- Suppose an optimal knapsack contaings x_1 kgs of item 1, x_2 kgs of item 2, $\cdots$, x_n kgs of item n, such that
$$x_1 + x_2 + \cdots + x_n = \min(w_{j^*}, W)$$
- We can replace this weight by $\min(w_{j^*}, W)$ kgs of item j^*
- Total weight does not change. Total value cannot decrease, because value per kg of j^* is maximum. So knapsack stays optimal

MST

Definitions

- A graph $G = (V, E)$ is a pair consisting of
  - A set V of vertices
  - A set $E \subseteq V \times V$ of edges
- $|E| = O(|V|^2)$
- Degree of a vertex v, $\deg(v)$ is the number of edges containing v
- Handshaking lemma: In any graph, $\sum_{v \in V} \deg(v) = 2 \cdot |E|$

Spanning tree

- A graph is connected if there is a path between any two vertices
- A tree is a connected graph without cycles
- A spanning tree of a connected graph $G = (V, E)$ is a tree that contains all of the vertices from V, and a subset of edges from E
- Given a weight function $w : E \to \mathbb{R}$, the weight of a spanning tree T is
$$\sum_{(u,v) \in T} w(u, v)$$
- A minimum spanning tree is a spanning tree with minimum weight

Optimal substructure

Any sub-tree T_k of the MST T is an MST of the subgraph induced by the vertices of T_k

Greedy-choice property

The least weight edge connecting any set of vertices to its complement is contained in the MST

MAX FLOW

Terminology

Flow network

- A flow network is a directed graph $G = (V, E)$ in which
  - If $(u, v) \in E$, then capacity $c(u, v) \geq 0$
  - If $(u, v) \notin E$, then capacity $c(u, v) = 0$
- For simplicity, no self-loops, no anti-parallel edges ($A \to B \to A$)

Flow

A flow is a function f that assigns a number $f(u, v)$ to each edge - the flow from u to v. It satisfies two properties:

1. Capacity: Flow $\leq$ Capacity
$$\forall (u, v) \in E, \quad 0 \leq f(u, v) \leq c(u, v)$$
2. Flow conservation: Flow in = Flow out
$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

Max flow

- The value of a flow f is
$$|f| = \sum_{v \in V} f(s, v)$$
assuming s has no incoming edges
- Max flow maximizes the value $|f|$ for a flow from start node s to destination node t

Bipartite matching

Problem

- Matching: A subset of edges such that each vertex is part of at most one edge
- Given a bipartite graph, what is the largest number of edges a matching can have

Solution

Bipartite matching reduces to max flow.

- Connect node s to all nodes in left half
- Connect all nodes in right half to node t
- Set all edges to have capacity 1

Need to show that the flow at each edge is either 1 or 0, to correspond to a bipartite matching solution. Running Ford-Fulkerson should guarantee this

Ford-Fulkerson

Formal algorithm

1. Start $f(u, v) = 0$ for all u, v
2. While there is a path p from s to t in G_f,
   - Let $m = \min_{(u,v) \in p} c_f(u, v)$
   - For each $(u, v) \in p$,
     - If $(u, v) \in E$, increment $f(u, v)$ by m
     - If $(v, u) \in E$, decrement $f(v, u)$ by m

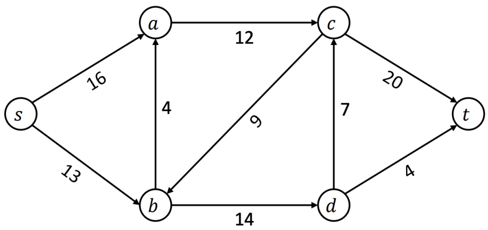
Residual capacities

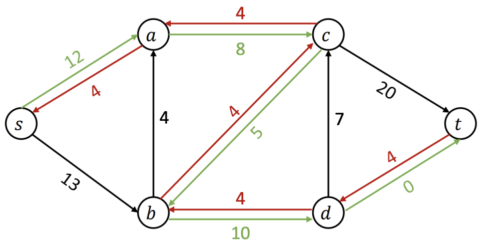
Given a flow f, define residual capacities c_f, which represents the remaining capacities:

- If $(u, v) \in E$, $c_f(u, v) = c(u, v) - f(u, v)$
- If $(v, u) \in E$, $c_f(u, v) = f(v, u)$
- Else, $c_f(u, v) = 0$

Residual graph

- Residual graph G_f keeps track of the capacities in "each direction"
- For example, if the path $s \to a \to c \to b \to d \to t$ is chosen, we can push a flow of 4 and get the following residual graph



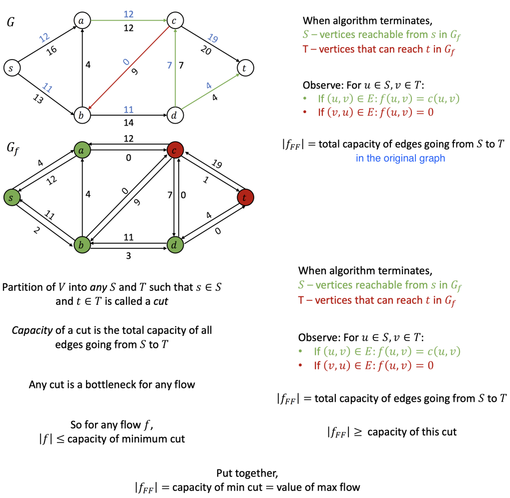


Instantiation

Ford-Fulkerson does not specify how the path should be found

- Find using DFS: $O(|E| \cdot |f_{max}|)$, where f_{max} is the maximal flow
- Edmonds-Karp algorithm - find (unweighted) shortest path using BFS: $O(|V||E|^2)$:

Max flow vs min cut



LINEAR PROGRAMMING

Definition

- Variables $x_1, \cdots, x_n$
- Maximize linear sum $\sum_{j \in [n]} c_j \cdot x_j$
- Subject to constraints which are linear inequalities:

$$x_1, \cdots, x_n \geq 0$$
$$a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1$$
$$a_{21}x_1 + \cdots + a_{2n}x_n \leq b_2$$
$$\cdots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m$$

- Any LP can be converted into this standard form
- Optimal solution (if exists) can be found in $poly(n, m)$ time

Converting to standard form

Minimize instead of maximize objective

Minimize $f(x) \implies$ Maximize $-f(x)$

Unbounded variables

If $x_1 \in \mathbb{R}$, then $x_1 = x_2 - x_3$, where $x_2, x_3 \geq 0$

Equality constraints

If $ax_1 + bx_2 = c$, then we can express this as two constraints:

$$-ax_1 + -bx_2 \leq -c$$
$$ax_1 + bx_2 \leq c$$

Sum of absolute values

Use linear programming to find a solution (a, b) that minimizes

$$\sum_{i=1}^{n} |y_i - ax_i - b|$$

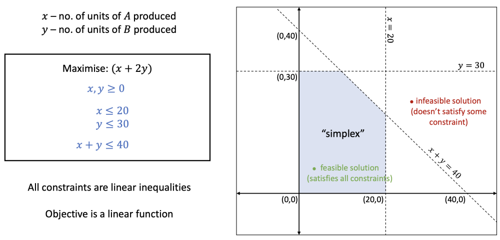
Define $e_i = |y_i - ax_i - b|$ such that

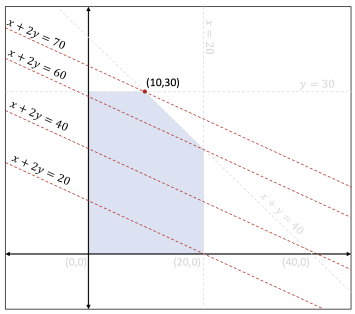
- $y_i - ax_i - b \leq e_i$
- $y_i - ax_i - b \geq -e_i$
- $e_i \geq 0$

Minimizing a max function

For minimizing $z = \max x_i$, define constraints $z \geq x_i$

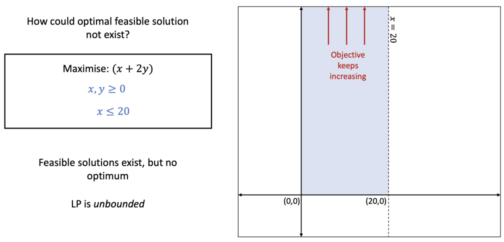
Visualizing



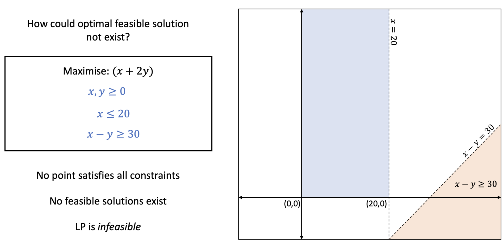


Optimal feasible solution DNE

LP is unbounded

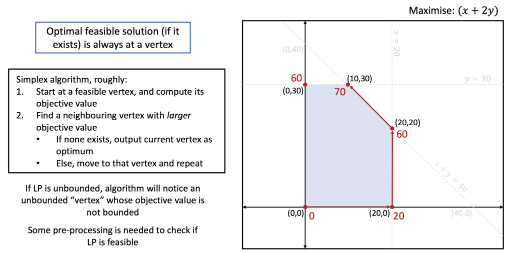


LP is infeasible



Simplex algorithm

- With n variables, the simplex is a n-dimensional convex polyhedron
- Convex $\implies$ correctness, since any local optimum is a global optimum
- Can take exponential time in worst case, but efficient in practice on typical inputs



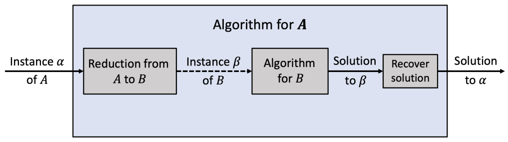
REDUCTIONS

Reductions

Consider two problems A and B. If A can be solved as follows, then we say A reduces to B.

Input: An instance α of A

1. Convert α into an instance β of B
2. Solve β and obtain a solution
3. Based on the solution of β, obtain the solution to α

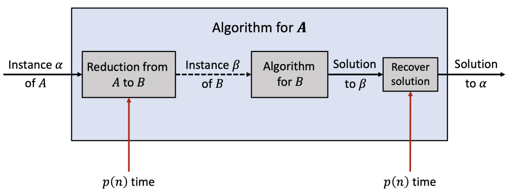


Bounded-time reduction

Consider two problems A and B, and a function p. If A can be solved as follows, then we say A has a $p(n)$-time reduction to B.

Input: An instance α of A of size n

1. Convert α into an instance β of B in $p(n)$ time
2. Solve β and obtain a solution
3. Based on the solution of β, obtain the solution to α in $p(n)$ time



Problem instance size

- n is the length of the encoding of the problem instance. It is roughly the number of bits used to write down that instance
- Can use standard encoding for many problems
  - Integers: Binary encoding
  - Graphs, matrices: List of parameters enclosed by braces, separated by commas

Running time

Running time composition

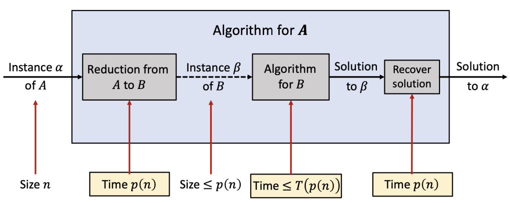
If there is

- a $p(n)$-time reduction from problem A to problem B, and
- a $T(n)$-time algorithm to solve problem B on instances of size n

then there is a

$$T(p(n)) + O(p(n))$$

time algorithm to solve problem A on instances of size n



Polynomial-time reduction

$A \leq_P B$ if there is a $p(n)$-time reduction from A to B for some polynomial function $p(n)$

- If B has a polynomial time algorithm, then so does A
- If A cannot be solved in polynomial time, then neither can B

Pseudo-polynomial algorithms

An algorithm that runs in time

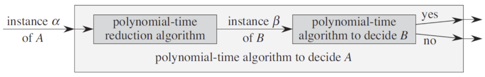
- polynomial in the numeric value of the input, but
- exponential in the length of the representation of the input

Intractability

Decision vs Optimization

- A decision problem is a function that maps each instance to either YES or NO
- Decision reduces to optimization. Given an instance of the optimization problem and a number k, ask whether there is a solution with value $\leq k$

Reductions between decision problems



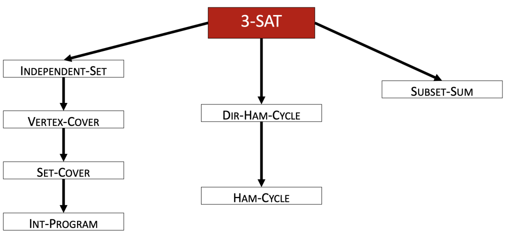
Suffices to show:

- Reduction runs in polynomial time
- If α is a YES-instance of A, β is a YES-instance of B
- If β is a YES-instance of B, α is a YES-instance of A

NP-COMPLETENESS

Reduction chart

Note the the arrow points in the direction of the reduction

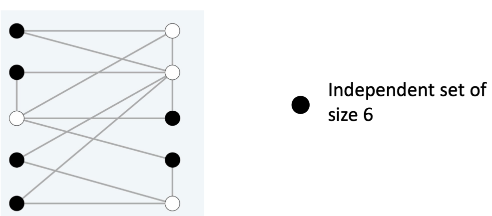


3-SAT is NP-hard and NP-complete, and so are the other problems in this chart.

Problem definitions

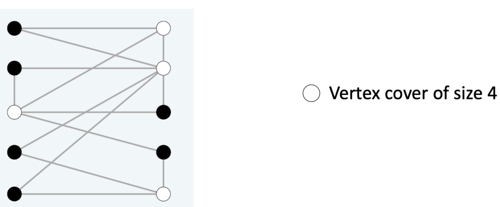
Independent-Set

Given a graph $G = (V, E)$ and an integer k, is there a subset of $\geq k$ vertices such that no two are adjacent?



Vertex-Cover

Given a graph $G = (V, E)$ and an integer k, is there a subset of $\leq k$ vertices such that each edge is incident to at least one vertex in the subset?



Set-Cover

Given integers k and n, and a collection $\mathcal{S}$ of subsets of $\{1, \cdots, n\}$, are there $\leq k$ of these subsets whose union equal $\{1, \cdots, n\}$?

$$k = 2, n = 7$$

$$S_1 = \{3,7\} \qquad S_4 = \{2,4\}$$
$$\boxed{S_2 = \{3,4,5,6\}} \qquad S_5 = \{5\}$$
$$S_3 = \{1\} \qquad \boxed{S_6 = \{1,2,6,7\}}$$

3-SAT

- Boolean variable: a variable that takes values True or False
- Literal: a Boolean variable or its negation
- Clause: a disjunction (OR) of literals, e.g. $C_j = x_1 \vee \bar{x}_2 \vee x_3$
- Conjunctive Normal Form (CNF) formula: a formula that is a conjunction (AND) of clauses
- Satifsying assignment: an assignment of x_i that makes the formula evaluate to True

Given a CNF formula ϕ over n variables, does it have a satisfying assignment?

Integer-Program

Given a set of m linear constraints in n variables

$$a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1$$
$$\cdots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m$$

is there an assignment of values $\{0, 1\}$ to the x_i such that all the constraints are satisifed

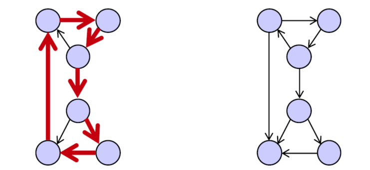
- Essentially a LP with additional constraint that each $x_i \in \{0, 1\}$

Subset-Sum

Given a list of integers S and a target t, decide if there is $S' \subseteq S$ that sums up to t.

Dir-Ham-Cycle

Given a directed graph $G = (V, E)$, is there a simple directed cycle Γ that contains every node in V exactly once?

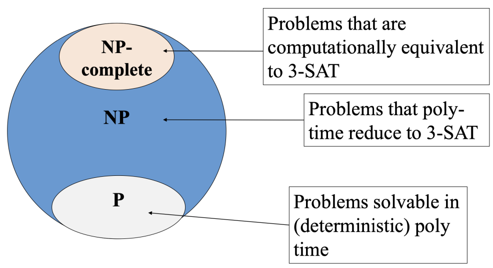


Has Hamiltonian cycle No Hamiltonian cycle

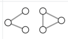
Ham-Cycle

Given an undirected graph $G = (V, E)$, is there a simple cycle that contains every node in V exactly once?

COMPLEXITY CLASSES



Some problems in P

problem	description	poly–time algorithm	yes	no
Multiple	Is x a multiple of y?	grade-school division	51, 17	51, 16
Rel-Prime	Are x and y relatively prime?	Euclid's algorithm	34, 39	34, 51
Primes	Is x prime?	Agrawal–Kayal–Saxena	53	51
Edit-Distance	Is the edit distance between x and y less than 5?	Needleman–Wunsch	niether neither	acgggt ttttta
L-Solve	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-Conn	Is an undirected graph G connected?	depth-first search		

NP

- Short for non-deterministic polynomial
- Defined as the class of problems for which polynomial time verifiable certificates of YES-instances exist
- There is a verification algo $V(x, y)$ that takes in an instance x and certificate y with $|y| = poly(|x|)$ such that
  $\exists y$ s.t. $V(x, y) = 1 \iff x$ is a YES-instance
- P $\subseteq$ NP because
  - Certificate can be anything
  - Verifier $V(x, \cdot)$ can solve for the instance x by itself and check if it is a YES-instance

Subset-Sum Certificate is the subset $S' \subseteq S$ that sums up to T. Verifier checks whether the sum of elements of S' is t, in polynomial time. Hence Subset-Sum is in NP.

co-NP

- A problem is in co-NP if polynomial time verifiable certificates of NO-instances exist
- The complement of any NP problem is in co-NP

NP-Hard

A problem A is said to be NP-Hard if for any problem B in NP,

$$B \leq_P A$$

Show NP-Complete

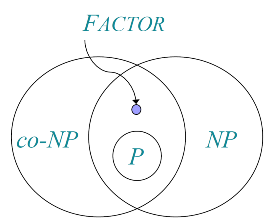
- Show that the problem is in NP, i.e. has a polynomial time verifiable certificate of YES-instance
- Show that the problem is in NP-hard, i.e. reduce FROM some known NP-hard problem TO this problem

Relationships

Cook-Levin Theorem

Any problem in NP poly-time reduces to 3-SAT. Hence, 3-SAT is NP-hard and NP-complete.

Likely relationships



Known:

- ❑ $P \subseteq (NP \cap co\text{-}NP)$
- ❑ $\text{Factor} \in (NP \cap co\text{-}NP)$

Hypothesized:

- ❑ $P \neq NP$
- ❑ $NP \neq co\text{-}NP$
- ❑ $\text{Factor} \notin P$

APPROXIMATION

Minimization

Given an instance, find a solution that has minimum cost.

- C^* - cost of optimal solution
- C - cost of solution found by your algorithm
- $\frac{C}{C^*}$ - approximation ratio, always larger than 1

Maximization

Given an instance, find a solution that has maximum cost.

- C^* - cost of optimal solution
- C - cost of solution found by your algorithm
- $\frac{C^*}{C}$ - approximation ratio, always larger than 1

PTAS

Polynomial-time approximation scheme

An algorithm that given an instance and $\epsilon > 0$, runs in time $poly(n)f(\epsilon)$ for some function f, and has approximation ratio $(1 + \epsilon)$

Fully PTAS

An algorithm that given an instance and $\epsilon > 0$, runs in time $poly(n, \frac{1}{\epsilon})$ for some function f, and has approximation ratio $(1 + \epsilon)$