Complexity

P: problems solvable in polynomial, \( n^c \), time.

EXP: problems solvable in exponential, \( 2^{n^c} \), time.

R: problems solvable in finite time.

Important relationship: \( P \subset EXP \subset R \)

Unsolvable problems

Halting Problem: Given a computer program, does it ever halt? (YES or NO decision problem): uncomputable, solvable in infinite time because you can just wait for the input to terminate (or halt), but no program can solve this problem in finite time.

Decision problems

Most decision problems are unsolvable.

You can represent a finite program as a finite binary string which can be mapped to a non-negative integer. The set of all programs representable by a finite binary string is the set of natural numbers \( \mathbb{N} \). You can represent a decision problem a function that maps a series of inputs to a result \( \in \{1, 0\} \), which represents \{YES, NO\}. The outputs of this function can be represented as an infinite binary string. We can represent the set of infinite binary strings by the set of real numbers, \( \mathbb{R} \), in the range \([0, 1]\).

\(|\mathbb{N}| << |\mathbb{R}|\)

So we cannot solve all decision problems using a finite program, in fact, we cannot solve most decision problems because \( \mathbb{R} \) is much larger than \( \mathbb{N} \).

P vs. NP

NP: decision problems with solutions that can be checked in polynomial time.

NP-hard: problems that are as hard as any problem in NP.

NP-complete: problems that are in NP and are NP-hard.

EXP-hard: problems that are as hard as any problem in EXP.

EXP-complete: problems that are in EXP and are EXP-hard.

The million dollar question of the century is whether \( P \) equals \( NP \).
Reductions

The goal of a reduction is to convert your problem into a problem that you already know how to solve. After turning your problem into the other problem, you can subsequently deduce the complexity of the unknown problem given you know the complexity of at least one of the two problems and that you perform the reduction in the correct direction.

\[ A \rightarrow B \]

This translates to "A reduces to B". From this reduction, we know that B is at least as hard as A.

Examples:

"unweighted shortest path" $\rightarrow$ "shortest weighted path with no negative edges"

Assume we don’t know how to solve "unweighted shortest path," but we know how to solve "shortest weighted path with no negative edges" with Dijkstra’s algorithm. We also know that Dijkstra’s algorithm $\in P$. We can transform the unweighted graph by making all edges weight 1. From this reduction, we know that finding "shortest weighted path with no negative edges" is at least as hard as "unweighted shortest path" so the problem of finding an "unweighted shortest path" $\in P$.

3-partition $\rightarrow$ Rectangle packing

Assume that we know that 3-partition (given n integers can you divide them into triples where the sum of each element in the triple is equal to the sum of every other triple) is NP-complete. Then, we can transform 3-partition into rectangle packing by creating a "box" of size $m \times \Sigma + 3m$ and rectangles of size $1 \times \Sigma + m$. We can set $\Sigma = \frac{1}{m}(a_1 + a_2 + \ldots + a_{3m})$ where $[a_1, a_2, \ldots, a_{3m}]$ are the n integers input into 3-partition. Then, we know that Rectangle packing returns "True" if and only if 3-partition returns "True." From this reduction, you have proven that "Rectangle packing" is at least as hard as "3-partition" so "Rectangle packing" is NP-hard.

Hamiltonian path $\rightarrow$ Hamiltonian cycle

Given that you know Hamiltonian path is NP-complete, then you can add a node to the graph input into the HAM-PATH problem and connect that node with every other node in the graph. Then, the transformed input into Hamiltonian cycle will return "True" if and only if the original input to Hamiltonian path returns "True." Thus, "Hamiltonian cycle" is NP-hard.

You can further prove that a problem is NP-complete after it has been proven to be NP-hard by showing that it is in the set NP.