02-613 Week 12

Algorithms and Advanced Data Structures

Aidan Jan

November 24, 2025

The Simplex Method

Suppose we have a system we want to optimize under some constraints. Say, maximize $z = x_1 + x_2$ under the constraints

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ x_1 - 3x_2 &\geq -1 \\ x_2 &\leq 3 \end{aligned}$$

Additionally, let all $x_i \geq 0$.

To use the simplex method, we first need to get these constraints into the **standard form**, which is to maximize $\vec{c} \cdot \vec{x}$, subject to $A\vec{x} \leq \vec{b}$ and $\vec{x} \geq 0$.

In our case, we need to do some rewriting. First, we have a greater than sign, which is not allowed. Therefore, we can convert it to a $\leq$ by multiplying both sides of the equation by -1.

$$\begin{aligned} x_1 + x_2 &\leq 3 \\ -x_1 + 3x_2 &\leq 1 \\ x_2 &\leq 3 \end{aligned}$$

Now, we add slack variables $S1, S2, S3$. The purpose of doing this is that [FILL rest of example]

The Simplex Algorithm

1. Write linear program with slack variables and get an initial feasible solution
2. Choose variable v in objective with positive coefficient.
3. Choose strictest constraint (will be one with negative coefficient on pivot)
4. Rewrite strictest constraint with v on the left hand side and subs in others
5. If all coefficients in the objective function are negative, then we are done. Otherwise, return to step 2.

Pivot Rules

- Largest coefficient in objective
- Largest increase after pivot
- Random

- Steepest change
- Bland's Rule - choose pivot v with lower index (This prevents infinite loops by always choosing the smallest index.)

NP Completeness

The phrase 'NP-Hard' tends to be used interchangeably with 'non-polynomial'. However, they are not the same thing.

Decision vs. Optimization Problems

- **Decision:** A problem that asks for a True/False solution.
- **Optimization:** A problem that asks for minimizing or maximizing some objective function.

Conveniently, any optimization problem can be turned into a decision problem with an extra parameter. For example:

- **Optimization:** Given graph $G = (V, E)$, and w weight on all edges, find the spanning tree with minimum total weight.
- **Decision:** Does there exist a spanning tree with at most $\leq k$ weight?

We refer to decision questions as 'problems'. Now we can define P and NP.

P vs. NP

Class P is a set of problems. A problem $Q \in P$, if there exists an algorithm A_Q such that, for all instances I of Q, $A_Q(I)$ runs in polynomial($|I|$) steps and

- If I is a yes instance $A_Q(I) =$ yes
- If I is a no instance $A_Q(I) =$ no

Class NP is a set of problems. A problem $Q \in NP$ if a **non-deterministic** Turing machine can return YES on a YES instance in polynomial time. A problem $Q \in NP$ if evidence of a Yes-instance can be verified in polynomial time. More formally, NP is a set of problems, Q for when there exists an algorithm $C_Q(\cdot)$ such that for all I of Q:

- If I is YES, then $C_Q(I, S) =$ Yes, for some S
- If I is NO, then $C_Q(I, S) =$ No, for all S
- Additionally, $C_Q(\cdot)$ runs in polynomial time.

This implies that problems can be both P *and* NP. $P \subset NP$.

Example: NP-Hard

Consider the traveling salesman decision problem. Given distances d_{ij} between n cities and a number k, is there a way to visit all cities exactly once, with total length of less than or equal to k?

- If given a valid certificate and a k-value (e.g., a YES instance), it is provable in polynomial time.
- Finding a valid certificate (e.g., an instance that satisfies the problem), is not possible though. Therefore, it is an NP-hard question.

Theorem: $P \subseteq NP$

Proof: Suppose $x \in P$ There exists a polynomial alorithm $A_x(\cdot)$. To show $X \in NP$, need efficient certifier $C_x(I, S) = A_x(I)$. Therefore, $P \subseteq NP$. Now, does $P = NP$? It turns out this is a hard problem in itself, and no proof exists for it.

Problem Reductions

A reduction from problem X to Y (written as $X \leq_P Y$), it means for instance I_X of X, create a new instance $(I_X)_Y$ in polynomial time, such that $\{(I_X)_Y$ is a yes instance $\longleftrightarrow I_X$ is a YES-instance$\}$, and Y is **at least as hard** as X.

- This implies that if there exists no polynomial time solution for X, then there must also not exist a polynomial time solution for Y.

Cook-Levin Theorem (1971)

This theorem states that every problem in NP can be reduced to SAT, or the satisfyability problem. The boolean satisfiability problem is NP-complete.

In turn, SAT can be reduced to 3SAT, which can be reduced to Independent set, vertex cover, set cover, hamiltonian path, TSP, graph coloring, 3D matching, and more.

The SAT problem

Suppose we have many boolean variables, $x_1, x_2, \ldots$, and a series of terms $t_1, t_2, \ldots$, where $t_i = x_i$ or $t_j = \bar{x_i}$. To combine terms, we have the two boolean operators, AND ($\wedge$) and OR ($\vee$). We write in conjunctive normal form, or CNF (AND-of-ORs, sum-of-products).

The problem is, if given an expression, e.g.

$$(x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee \bar{x_3}) \wedge (x_2 \vee x_3)$$

does there exist a truth assignment (choice of T/F for each variable) such that the entire expression returns true? A **satisfying assignment** is a truth assignment that makes the formula true.

This problem is NP-complete.

The 3SAT problem

The 3SAT problem is a subset of SAT, where every OR clause contains exactly three terms. We care about this specific subset because as it turns out, every SAT problem is reducable to 3SAT. How is this possible?

There are three cases in converting a SAT problem to a 3SAT problem. Either the clause in SAT has less than 3 terms, exactly 3 terms, or more than 3 terms.

- If there are exactly three terms, then nothing changes. This case already meets the conditions for 3SAT.
- If there are less than three terms, then duplicate a variable. e.g., $(t_1 \vee t_2) = (t_1 \vee t_2 \vee t_2)$.
- If there are more than three terms, this process is less trivial. To do this case, we add variables in the following way:

$$\begin{aligned}
&(t_1 \vee t_2 \vee t_3 \vee t_4 \vee \cdots \vee t_k) \\
=\ &(t_1 \vee t_2 \vee Y_1) \wedge \\
&\qquad\qquad (\bar{Y_1} \vee t_3 \vee Y_2) \wedge \\
&\qquad\qquad\qquad\qquad (\bar{Y_2} \vee t_4 \vee Y_3) \wedge \\
&\qquad\qquad\qquad\qquad\qquad \cdots \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\bar{Y}_{k-1} \vee t_{k-1} \vee t_k)
\end{aligned}$$

Doing this reduction keeps the same statement as the original, but now every clause only contains three terms.

Proving 3SAT NP-Hard

Now, we need to prove 3SAT is NP-Hard in order to be able to use it in other NP-Hard proofs. What we can do is convert this problem into an independent set problem. To do this, draw a graph. Each clause of three terms OR'ed together is converted to 3 nodes on the graph connected with edges. Furthermore, each node that reference the same term (e.g., if t_1 is in two different clauses), the two nodes are also connected with an edge. Now, this is an independent set problem, where whichever nodes are set to true are possible values that would satisfy 3SAT.