Computational Complexity in Numerical Optimization

ORF 523
Lecture 13

Instructor: Amir Ali Ahmadi

Spring 2018
When can we solve an optimization problem efficiently?

Arguably, this is the main question an optimizer should be concerned with. At least if his/her view on optimization is computational.

A quote from our first lecture:

- The right question in optimization is **not**
  - Which problems are optimization problems?
    (The answer would be everything.)

- The right question is
  - *Which optimization problems can we solve?*
Is it about the number of decision variables and constraints?

▪ No!

▪ Fermat’s conjecture (1637):
For \( n \geq 3 \), the equation \( x^n + y^n = z^n \) does not have a solution over positive integers.

▪ Consider the following optimization problem:

\[
\begin{align*}
\text{min.} & \quad (x^n + y^n - z^n)^2 \\
\text{s.t.} & \quad x \geq 1, \quad y \geq 1, \quad z \geq 1, \quad n \geq 3,
\end{align*}
\]

\[
\sin^2 \pi n + \sin^2 \pi x + \sin^2 \pi y + \sin^2 \pi z = 0.
\]

▪ Innocent-looking optimization problem: 4 variables, 5 constraints.

▪ If you could show the optimal value is non-zero, you would prove Fermat’s conjecture!
Is it about being linear versus nonlinear?

- No!

- We can solve many nonlinear optimization problems efficiently:
  - QP
  - Convex QCQP
  - SOCP
  - SDP
  - ...

Is it about being convex versus nonconvex?

Hmmm...many would say

(if I could only get 10 cents for every time somebody gave a talk and justified giving a heuristic by saying that his problem was nonconvex....)

Famous quote -
Rockafellar, ’93:

“In fact the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity.”

But is it really?
We already showed that you can write any optimization problem as a convex problem:

- Write the problem in epigraph form to get a linear objective
- Replace the constraint set with its convex hull

So at least we know it’s not just about the geometric property of convexity; somehow the (algebraic) description of the problem matters

There are many convex sets that we know we cannot efficiently optimize over

- Or we cannot even test membership to
- We’ll see some examples (set of copositive matrices, positive polynomials, etc.).
Is it about being convex versus nonconvex?

- Even more troublesome, there are non-convex problems that are easy.
- Who can name four of them that we’ve seen already?

1. \[
\min_{\mathbf{x}} \mathbf{x}^T \mathbf{B} \mathbf{x} \\
\text{st. } \mathbf{x}^T \mathbf{x} = 1
\]

2. \[
\min_{\mathbf{x}} \mathbf{x}^T \mathbf{Q}_1 \mathbf{x} + \mathbf{b}_1^T \mathbf{x} + c_1 \\
\text{st. } \mathbf{x}^T \mathbf{Q}_2 \mathbf{x} + \mathbf{b}_2^T \mathbf{x} + c_2 \leq 0
\]

(The S-lemma)
Is it about being convex versus nonconvex?

- Even more troublesome, there are non-convex problems that are easy.
- Who can name four of them that we’ve seen already?

\[
\begin{align*}
\text{min}_B & \quad \|A-B\|_F \\
\text{rank}(B) & \leq K \\
\text{(SVD)}
\end{align*}
\]

\[
\begin{align*}
\text{State feedback stabilization} \\
\text{Given } A, B, \exists K \text{ s.t.} \\
\rho(A+BK) & < 1 \\
\text{(LMI tricks)}
\end{align*}
\]
Is it about being convex versus nonconvex?

- I admit the question is tricky
- For some of these non-convex problems, one can come up with an equivalent convex formulation
- But how can we tell when this can be done?
- We saw, e.g., that when you tweak the problem a little bit, the situation can change
  - Recall, e.g., that for output feedback stabilization we had no convex formulation
  - Or for generalization of the S-lemma to QCQP with more constraints...
- Can we have techniques for showing that (an efficiently solvable) convex formulation is impossible?
Is it about being convex versus nonconvex?

- My view on this question:
  - Convexity is a rule of thumb.
  - It’s a very useful rule of thumb.
    - Often it characterizes the complexity of the problem correctly.
    - But there are exceptions.
  - Incidentally, it may not even be easy to check convexity unless you are in pre-specified situations (recall the CVX rules for example).
    - Maybe good enough for many applications.
  - To truly and rigorously speak about complexity of a problem, we need to go beyond this rule of thumb.
  - Computational complexity theory is an essential tool for optimizers.
Why computational complexity?

- What is computational complexity theory?
  It’s a branch of mathematics that provides a formal framework for studying how efficiently one can solve problems on a computer.

- This is absolutely crucial to optimization and many other computational sciences.
  - In optimization, we are constantly looking for algorithms to solve various problems as fast as possible. So it is of immediate interest to understand the fundamental limitations of efficient algorithms.

- To start, how can we formalize what it means for a problem to be “easy” or “hard”?
  - Let’s begin by understanding what it means to have a “problem”!
Let’s introduce these concepts using an example we know well: stable set (aka independent set) of a graph.

Recall that a stable set in a graph $G$ is a subset of the nodes with no edges among them.

Optimization problem:
Given a graph $G$, find its largest stable set.

Decision problem:
Given a graph $G$ and an integer $b$, decide if there exists a stable set of size $\geq b$?
(answer to a decision question is just YES or NO)

Search problem:
Given a graph $G$ and an integer $b$, find a stable set of size $\geq b$ or declare that none exists.

It turns out that all three problems are equivalent, in the sense that if you could solve one efficiently, you could also solve the other two (why?). See Ex. 8.1,8.2 of [DPV].

We will focus on decision problems, since it’s a bit cleaner to develop the theory there, and since it can only make our negative results stronger.
A “problem” versus a “problem instance”

- A (decision) problem is a general description of a problem to be answered with yes or no.
- Every decision problem has a *finite input* that needs to be specified for us to choose a yes/no answer.
- Each such input defines an **instance** of the problem.
- A decision problem has an infinite number of instances. (Why doesn’t it make sense to study problems with a finite number of instances?)

Different instances of the STABLE SET problem:
(It is common to use capital letters for the name of a decision problem.)
Examples of decision problems

**LINEQ**

**Input:** An $m \times n$ matrix $A$ and an $m \times 1$ vector $b$, both with rational entries.

**Question:** Is there a solution to the linear system $Ax = b$?

An instance of LINEQ:

$$
2x_1 + 7x_2 = 6 \\
\frac{1}{2}x_1 - x_2 = -\frac{1}{3}
$$

$$
A = \begin{pmatrix}
2 & 7 \\
\frac{1}{2} & -1
\end{pmatrix}, \quad b = \begin{pmatrix}
6 \\
-\frac{1}{3}
\end{pmatrix}
$$

**ZOLINEQ**

**Input:** An $m \times n$ matrix $A$ and an $m \times 1$ vector $b$, both with rational entries.

**Question:** Is there a 0/1 solution $x$ to the linear system $Ax = b$?

An instance of ZOLINEQ:

$$
2x_1 + 7x_2 = 6 \\
\frac{1}{2}x_1 - x_2 = -\frac{1}{3}
$$

$$
A = \begin{pmatrix}
2 & 7 \\
\frac{1}{2} & -1
\end{pmatrix}, \quad b = \begin{pmatrix}
6 \\
-\frac{1}{3}
\end{pmatrix}
$$

**Remark.** Input is rational so we can represent it with a finite number of bits. This is the so-called “bit model of computation”, aka the “Turing model.”
Examples of decision problems

**LP**

**Input:** An $m \times n$ matrix $A$, an $m \times 1$ vector $b$, and an $n \times 1$ vector $c$, all rational

a rational number $k$

**Question:** Is the optimal value of the LP (in standard form) $\leq k$?

(This is equivalent to testing LP feasibility.)

An instance of LP: 

$$A = \begin{pmatrix} 4 & 7 & \frac{1}{2} \\ \frac{3}{2} & -1 & \frac{3}{2} \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad k = 5.$$ 

**IP**

**Input:** same as above

**Question:** Is there an integer feasible solution to the LP with objective value $\leq k$?
Examples of decision problems

Let’s look at a problem we have seen...

An instance of MAXFLOW:

Can you formulate the decision problem?

**MAXFLOW**

**Input:** A directed graph $G(V, E)$, nonnegative rational numbers $c_i$ on each edge, a designated node $S$, a designated node $T$, a rational number $k$.

**Question:** Is there a flow of value $\geq k$ from $S$ to $T$ that respects the edge cost constraints and the conservation of flow constraints?
Examples of decision problems

- A graph is said to be $k$-colorable if there is a way to color its nodes with $k$ colors such that no two adjacent nodes get the same color.

- For example, the following graph is 3-colorable.

- Graph coloring has important applications in job scheduling.

**COLORING**

**Input:** An undirected graph $G$ and a positive integer $k$.

**Question:** Is the graph $k$-colorable?

- We want to understand how fast can all these problems be solved?
To talk about the running time of an algorithm, we need to have a notion of the “size of the input”.

Of course, an algorithm is allowed to take longer on larger instances.

Reasonable candidates for input size:

- Number of nodes $n$
- Number of nodes + number of edges (number of edges can at most be $n(n-1)/2$)
- Number of bits required to store the adjacency matrix of the graph
Size of an instance

- In general, can think of input size as the total number of bits required to represent the input.

- For example, consider our LP problem:

- **LP**

- **Input:** An $m \times n$ matrix $A$, an $m \times 1$ vector $b$, and an $n \times 1$ vector $c$, all rational
  a rational number $k$

- **Question:** Is the optimal value of the LP (in standard form) $\leq k$?

  - Input size is bounded by $2(mn + m + n + 1) \log L$, where $L$ is the largest
    integer appearing in the numerator or denominator of any entry of $A, b, c, k$.

  - Same idea holds for all other decision problems we introduced.
Useful notation for referring to running times

**Definition.** Let \( f, g : \mathbb{R}_+ \rightarrow \mathbb{R}_+ \). We write

- \( f(n) = O(g(n)) \), if \( \exists n_0, c > 0 \), such that \( f(n) \leq cg(n), \forall n \geq n_0 \).

- \( f(n) = \Omega(g(n)) \), if \( \exists n_0, c > 0 \), such that \( f(n) \geq cg(n), \forall n \geq n_0 \).

- \( f(n) = \Theta(g(n)) \), if we have both \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \).

**Examples.**

- \( 5n^3 + 2n^2 + 40 = \Theta(n^3) \).
- \( n \log n = O(n^2) \).
- \( n \log n = \Omega(n) \).
- \( \forall c, k > 0, 2^{cn} = \Omega(n^k) \).
Polynomial-time and exponential-time algorithms

- A polynomial-time algorithm is an algorithm whose running time as a function of the input size is \( O(p(n)) \) for some polynomial function \( p \).

- Equivalent definition: Running time is \( O(n^k) \) for some positive integer \( k \).

- Note: this is the worst-case running time over all inputs of size \( n \).

- An exponential-time algorithm is an algorithm whose running time as a function of the input size is \( \Omega(2^{cn}) \) for some positive constant \( c \).

- Once again, when we talk about running time for a given input size \( n \), we mean the worst-case running time over all inputs of size \( n \).

- There are also algorithms with running time in between (e.g., \( O(n^\log n) \)), but these also are perceived as slow.
## Comparison of running times

<table>
<thead>
<tr>
<th>Time complexity function</th>
<th>Size $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$n$</td>
<td>.00001 second</td>
</tr>
<tr>
<td>$n^2$</td>
<td>.00001 second</td>
</tr>
<tr>
<td>$n^3$</td>
<td>.001 second</td>
</tr>
<tr>
<td>$n^5$</td>
<td>.1 second</td>
</tr>
<tr>
<td>$2^n$</td>
<td>.001 second</td>
</tr>
<tr>
<td>$3^n$</td>
<td>.059 second</td>
</tr>
</tbody>
</table>

Image credit: [GJ79]
Can Moore’s law come to rescue?

<table>
<thead>
<tr>
<th>Time complexity function</th>
<th>With present computer</th>
<th>With computer 100 times faster</th>
<th>With computer 1000 times faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$N_1$</td>
<td>$100 \ N_1$</td>
<td>$1000 \ N_1$</td>
</tr>
<tr>
<td>$n^2$</td>
<td>$N_2$</td>
<td>$10 \ N_2$</td>
<td>$31.6 \ N_2$</td>
</tr>
<tr>
<td>$n^3$</td>
<td>$N_3$</td>
<td>$4.64 \ N_3$</td>
<td>$10 \ N_3$</td>
</tr>
<tr>
<td>$n^5$</td>
<td>$N_4$</td>
<td>$2.5 \ N_4$</td>
<td>$3.98 \ N_4$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$N_5$</td>
<td>$N_5 + 6.64$</td>
<td>$N_5 + 9.97$</td>
</tr>
<tr>
<td>$3^n$</td>
<td>$N_6$</td>
<td>$N_6 + 4.19$</td>
<td>$N_6 + 6.29$</td>
</tr>
</tbody>
</table>

Effect of improved technology on several polynomial and exponential time algorithms.

Image credit: [GJ79]
The complexity class P

- The class of all decision problems that admit a polynomial-time algorithm.

- ADDITION
- MULTIPLICATION
- LINEQ
- LP
- MAXFLOW
- MINCUT
- MATRIXPOS
- SHORTEST PATH
- SDP $\epsilon$
- PRIMES
- ZEROSUMNASH
- PENONPAPER,...
Example of a problem in P

**PENONPAPER**

**Input:** A connected undirected graph.

**Question:** Can you draw it without lifting your pen from the paper?

![Graphs](image)

**Euler:** answer to PENONPAPER is YES if and only if “every node, with the possible exception of two nodes, has even degree.”

This condition can obviously be checked in polynomial time.

Hence PENONPAPER $\in \mathbb{P}$.

**Peek ahead:** this problem is asking if there is a path that visits *every edge exactly once*.

If we were to ask for a path that instead visits *every node exactly once*, we would have a completely different story in terms of complexity!
How to prove a problem is in P?

- Develop a poly-time algorithm from scratch! Can be far from trivial (examples below).
- Much easier: use a poly-time hammer somebody else has developed. (Reductions!)

**LINEQ** (solve a system of linear equations)

- Gaussian elimination -- $O(n^3)$
- Can also use, e.g., the conjugate gradient algorithm -- $O(n^3)$
- (Faster algorithms known: Google Strassen)

**LP** (solve a system of linear inequalities)

- Was open for a long time – simplex doesn’t do it (at least, we don’t know how to modify it so it does)
- The ellipsoid algorithm (Khachiyan-1979)
- Interior point algorithms (Karmarkar-1984)

**PRIMES** (decide if a given integer is prime)

- Was open for a long time -- Proved to be in P by Agrawal-Kayal-Saxena in 2002.
  - Kayal and Saxena were undergraduates!
- Why doesn’t the naïve algorithm work? “Given $n$, check all candidate divisors up to $\sqrt{n}$.”
An aside: Factoring

▪ Despite knowing that PRIMES is in P, it is a major open problem to determine whether we can \textit{factor} an integer in polynomial time.

$\text{RSA-1024} = 13506641086599522334960321627880596993888147560566702752448514385152651060$
$485953383394028715057190941792072821644715513736804197093646419174304646965$
$89274256239341020864383202110372958725762358509643110564073501508187510676$
$5946292055636855294752135008528794163773285339061097505433499981115005697$
$7236890927563$

$\text{RSA-2048} = 2519590847565789349402718324004839857142928216260430320777713783604366202070$
$75955562640185258807844069182906412495150812892985591491761814502808489120072$
$8449926873928072787776735971418347270261896375014971824691165076133798590957$
$0009733045974880842840179742910064245869181795118764121515172654632282221686$
$99875491824224336372590851418654620435676984233871847774447920739934236584823$
$8242811981638150106748104516603730650620161966256133844143603838904419526$
$34432190114657544454178424020921563235077870774981712577246796292638635$
$637328991215483134816798859004044565042537834321200397122822$
$120720357$

$\$100,000 prize money by RSA$

$\$200,000 prize money by RSA$

▪ Google “RSA challenge”; was active until 2007.
Reductions

- Many new problems are shown to be in P via a reduction to a problem that is already known to be in P.

- What is a reduction?
  - Very intuitive idea -- A reduces to B means: “If we could do B, then we could do A.”
  - Being happy in life reduces to finding a good partner.
  - Passing the quals reduces to getting four A-’s.
  - Getting an A+ in ORF 523 reduces to finding the Shannon capacity of C7.

- Well-known joke - mathematician versus engineer boiling water:

  - Day 1:

  - Day 2:
A reduction from a decision problem A to a decision problem B is

- a “general recipe” (aka an algorithm) for taking any instance of A and explicitly producing an instance of B, such that
  - the answer to the instance of A is YES if and only if the answer to the produced instance of B is YES.

- This enables us to answer A by answering B.
MAXFLOW $\rightarrow$ LP

**MAXFLOW**

**Input:** A directed graph $G(V, E)$, nonnegative rational numbers $c_i$ on each edge, a designated node $S$, a designated node $T$, a rational number $k$.

**Question:** Is there a flow of value $\geq k$ from $S$ to $T$ that respects the edge cost constraints and the conservation of flow constraints?

**LP**

**Input:** An $m \times n$ matrix $A$, an $m \times 1$ vector $b$, and an $n \times 1$ vector $c$, all rational, a rational number $k$.

**Question:** Is the optimal value of the LP $\geq k$?

---

Poly-time reduction (shown on once instance)

\[
\begin{align*}
\text{max.} & \quad x_{SA} + x_{SB} + x_{SC} \\
\text{s.t.} & \quad x_{SA} + x_{AD} + x_{AE} + \cdots + x_{GT} \leq k \\
& \quad x_{SA} \leq 6, \quad x_{SB} \leq 2, \quad x_{SC} \leq 10, \quad \ldots, \quad x_{GT} \leq 12 \\
& \quad x_{SA} = x_{AD} + x_{AB} + x_{AE} \\
& \quad x_{SC} = x_{EB} + x_{CF} \\
& \quad \vdots \\
& \quad x_{CF} + x_{EF} = x_{TF}.
\end{align*}
\]
Polynomial time reductions

- So we say that “MAXFLOW reduces to LP”. (Notation: MAXFLOW $\rightarrow$ LP.)

- Since we know how to solve LP in polynomial time (e.g., via interior point methods), now we know how to solve MAXFLOW in polynomial time. So MAXFLOW $\in$ P.

- This argument relies crucially on the fact that the reduction is polynomial in length.

  - Before we even solve the LP, we need to make sure its size is not too big (e.g., it doesn’t have too many decision variables, too many constraints, or data that takes an exponential number of bits to write down.)

  - What does “not too big” mean? The size needs to be polynomial in the size of the instance of the original problem (in this case MAXFLOW).

  - Without this constraint, one could give, e.g., a simple reduction from STABLE SET to LP (do you see how)? This should not happen (we’ll see why soon).
- **MINCUT**

  - **A cut** is a partition of the nodes of a graph into two (non-empty) sets $U$ and $\overline{U}$.
  - The **value of a cut** is the sum of edge weights going from $U$ to $\overline{U}$.

**MINCUT**

**Input:** A directed graph $G(V,E)$, nonnegative rational numbers $c_i$ on each edge, a rational number $k$.

**Question:** Is there a cut of value $\leq k$?

**Is MINCUT in P?**

- Yes! We’ll reduce it to LP.
**MIN S-T CUT**

**Input:** A directed graph $G(V, E)$, nonnegative rational numbers $c_i$ on each edge, a rational number $k$, two designated nodes $S$ and $T$.

**Question:** Is there a cut of value $\leq k$? (that leaves $S$ on one side and $T$ on the other)

**Strong duality** of linear programming implies the minimum S-T cut of a graph is exactly equal to the maximum flow that can be sent from $S$ to $T$.

Hence, **MIN S-T CUT $\rightarrow$ MAXFLOW**

We have already seen that **MAXFLOW $\rightarrow$ LP**.

But what about **MINCUT**? (without designated $S$ and $T$)
MINCUT $\Rightarrow$ MIN S-T CUT

- Pick a node (say, node A)
- Compute MIN S-T CUT from A to every other node
- Compute MIN S-T CUT from every other node to A
- Take the minimum over all these $2(|V|-1)$ numbers
- That’s your MINCUT!
- The reduction is polynomial in length.
Overall reduction

- We have shown the following:

\[ \text{MINCUT} \rightarrow \text{MIN S-T CUT} \rightarrow \text{MAXFLOW} \rightarrow \text{LP} \]

- Polynomial time reductions compose (why?):

\[ \text{MINCUT} \rightarrow \text{LP} \]

- \text{MINCUT} \in \text{P}

- Unfortunately, we are not so lucky with all decision problems...

- Now comes the bad stuff...
MAXCUT

Input: A graph $G(V, E)$, nonnegative rational numbers $c_i$ on each edge, and a rational number $k$.

Question: Is there a cut of value $\geq k$?

Examples with edge costs equal to 1:

- Cut value = 8
- Cut value = 23 (optimal)

To date, no one has come up with a polynomial time algorithm for MAXCUT.

We want to understand why that is...
The traveling salesman problem (TSP)

**TSP**

**Input:** A graph $G(V, E)$, nonnegative rational numbers $c_i$ on each edge, a rational number $k$.

**Question:** Is there a tour of cost $\leq k$ that visits each node exactly once?

Again, nobody knows how to solve this efficiently (over all instances).

Note the sharp contrast with PENONPAPER.

Amazingly, MAXCUT and TSP are in a precise sense “equivalent”: there is a polynomial time reduction between them in either direction.
TSP

24,978 Cities in Sweden
Solved in 2004

15,112 Cities in Germany
Solved in 2001

Reference: http://www.math.uwaterloo.ca/tsp
The complexity class NP

- A decision problem belongs to the class **NP (Nondeterministic Polynomial time)** if every YES instance has a “certificate” of its correctness that can be verified in polynomial time.

- Examples: TSP, MAXCUT, PENONPAPER....what’s the certificate in each case?

**Remarks.**

- A nondeterministic computer is a machine that can “guess” an answer and then verify it. It’s a very unrealistic computer.

- NP does not mean “not polynomial”! There are many easy problems in NP (e.g., ADDITION, LINEQ).

- P ⊆ NP. (The poly-time algorithm itself is a certificate.)

- Note that for a given decision problem, it’s not at all clear that a short certificate for the YES answer also implies a short certificate for the NO answer. (Think, e.g., of TSP.)
The complexity class NP

- ADDITION
- MULTIPLICATION
- LINEQ
- LP
- MAXFLOW
- MINCUT
- MATRIXPOS
- SHORTEST PATH
- SDP $\epsilon$
- PRIMES
- ZEROSUMNASH
- PENONPAPER,...

- TSP
- MAXCUT
- STABLE SET
- SAT
- 3SAT
- PARTITION
- KNAPSACK
- IP
- COLORING
- VERTEXCOVER
- 3DMATCHING
- SUDOKU,...
NP-hard and NP-complete problems

Definition.

A decision problem is said to be **NP-hard** if every problem in NP reduces to it via a polynomial-time reduction.
(roughly means “harder than all problems in NP.”)

Definition.

A decision problem is said to be **NP-complete** if

(i) It is NP-hard

(ii) It is in NP.

(roughly means “the hardest problems in NP.”)

Remarks.

NP-hardness is shown by a reduction from a problem that’s already known to be NP-hard.

Membership in NP is shown by presenting an easily checkable certificate of the YES answer.

NP-hard problems may not be in NP (or may not be known to be in NP as is often the case.)
The complexity class NP

- ADDITION
- MULTIPLICATION
- LINEQ
- LP
- MAXFLOW
- MINCUT
- MATRIXPOS
- SHORTEST PATH
- SDP_\epsilon
- PRIMES
- ZEROSUMNASH
- PENONPAPER,...

\( \in \mathbb{P} \)

- TSP
- MAXCUT
- STABLE SET
- SAT
- 3SAT
- PARTITION
- KNAPSACK
- IP
- COLORING
- VERTEXCOVER
- 3DMATCHING
- SUDOKU,...

NP-complete
The satisifiability problem (SAT)

- **SAT** (one of the most fundamental NP-complete problems.)

- **Input:** A Boolean formula in conjunctive normal form (CNF).

- **Question:** Is there a 0/1 assignment to the variables that satisfies the formula?

\[ \varphi = (x \lor y \lor z) \land (x \lor \overline{y}) \land (y \lor \overline{z}) \land (\overline{x} \lor \overline{y} \lor \overline{z}) \]

**Formula**

**Clauses**

**Variables:** $x, y, z$

$\lor$: OR, $\land$: AND, $\overline{x}$: NOT $x$

**Literal:** a variable or its complement.

\[
\begin{array}{c|c|c}
\text{AND} & \text{X} & \text{X} \\
\hline
A & B & A \cdot B \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{OR} & \text{X} & \text{X} \\
\hline
A & B & A +B \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
The satisfiability problem (SAT)

- **Input**: A Boolean formula in conjunctive normal form (CNF).
- **Question**: Is there a 0/1 assignment to the variables that satisfies the formula?

\[ (x \lor y \lor z) \land (x \lor \overline{y}) \land (y \lor \overline{z}) \land (\overline{x} \lor \overline{y} \lor \overline{z}) \]

**YES** \( x=1, y=1, z=0 \).

\[ (x \lor y \lor z) \land (x \lor \overline{y}) \land (y \lor \overline{z}) \land (\overline{x} \lor \overline{y} \lor \overline{z}) \land (\overline{x} \lor z) \]

**NO**
3SAT

Input: A Boolean formula in conjunctive normal form (CNF), where each clause has exactly three literals.

Question: Is there a 0/1 assignment to the variables that satisfies the formula?

\[(\chi \lor \bar{y} \lor z) \land (\bar{x} \lor y \lor \bar{z}) \land (x \lor \omega \lor \bar{z}) \land (\bar{y} \lor \omega \lor z)\]

There is a simple reduction from SAT to 3SAT.

Hence, since SAT is NP-hard, then so is 3SAT. Moreover, 3SAT is clearly in NP (why?), so 3SAT is NP-complete.
Reductions (again)

- A reduction from a decision problem A to a decision problem B is
  - a "general recipe" (aka an algorithm) for taking any instance of A and explicitly producing an instance of B, such that
    - the answer to the instance of A is YES if and only if the answer to the produced instance of B is YES.

  - This enables us to answer A by answering B.

- This time we use the reduction for a different purpose:
  - If A is known to be hard, then B must also be hard.
Today we have thousands of NP-complete problems. In all areas of science and engineering.
The value of reductions

I can’t find an efficient algorithm, I guess I’m just too dumb.

I can’t find an efficient algorithm, because no such algorithm is possible.

I can’t find an efficient algorithm, but neither can all these famous people.

[Garey, Johnson]
Practice with reductions

I’ll do a few reductions on the board:

- **3SAT \rightarrow** STABLE SET
- **STABLE SET \rightarrow** 0/1 IP (trivial)
- **STABLE SET \rightarrow** QUADRATIC EQS (trivial)
- **3SAT \rightarrow** POLYPOS (degree 6)
- **ONE-IN-THREE 3SAT \rightarrow** POLYPOS (degree 4)
- NP-hardness of testing local optimality

For homework you can do:

- **3SAT \rightarrow** ONE-IN-THREE 3SAT
- **PARTITION \rightarrow** POLYPOS (degree 4)
3SAT $\rightarrow$ STABLE SET

We show the reduction on an instance only. The pattern should be clear.

$$\Phi = (\bar{x} \lor y \lor \bar{z}) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor y \lor \bar{z}) \land (x \lor \bar{y} \lor \bar{z})$$

($k$ clauses)

**Construction:** For each clause create a "triangle." Across triangles, connect each variable to its complement.

**Claim:** $\Phi$ is satisfiable $\iff \alpha(G) \geq k$. 

[Diagram of a graph with variables $x, y, z$ connected in a specific pattern to illustrate the construction.]
STABLE SET \rightarrow 0/1 Integer Programming

Given \( G(V,E) \)

\[ \alpha(G) \geq k \]

\[ \Leftrightarrow \]

\[ \left\{ \begin{array}{l}
\sum_{i=1}^{n} x_i \geq k \\
x_i + x_j \leq 1 \quad \text{if } ij \in E \\
x_i \in \{0,1\} \quad i = 1, \ldots, n
\end{array} \right. \]

is feasible.
STABLE SET $\Rightarrow$ Feasibility of Quadratic Equations

Given $G(V,E)$

$\alpha(G) \geq k$

$\iff$

$\begin{cases}
\sum_{i=1}^{n} x_i - k = s^2 \\
x_i x_j = 0 \quad \text{if } i,j \in E \\
x_i (1-x_i) = 0 \quad i = 1, \ldots, n
\end{cases}$

is feasible.
3SAT $\Rightarrow$ POLYPOS (degree 6)

We show the reduction on an instance only. The pattern should be clear.

Start with any instance of 3SAT, such as:

$$Q = (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor x_4)$$

Construct $p$ as

$$p(x) = \sum_{i=1}^{4} (x_i (1-x_i))^2 + \left[ (x_1+(1-x_2)+x_3-1) (x_1+(1-x_2)+x_3-2) (x_1+(1-x_2)+x_3-3) \right]^2$$

Observe that the reduction is polynomial in length.
Claim: A general instance \( \Phi \) of 3SAT will be satisfiable

\[ \exists \bar{x} \in \mathbb{R}^n \text{ such that } p(\bar{x}) \leq 0 \text{ (in fact } p(\bar{x}) = 0), \text{ where } p \text{ is constructed as above.} \]

\( \text{Pf.} \) (\( \uparrow \)) Take \( \bar{x} \) to be the satisfying assignment of 3SAT.

All the terms of \( p \) vanish (why?)

\( \downarrow \) Suppose \( \Phi \) not satisfiable. Claim: \( p(x) > 0 \ \forall x \in \mathbb{R}^n \).

\( p \) is a sum of squares \( \Rightarrow p(x) > 0 \ \forall x \in \mathbb{R}^n \).

a) If \( x \notin \{0, 1\}^n \), then \( \sum (x_i (1-x_i))^2 > 0 \) (why?)

b) If \( x \in \{0, 1\}^n \), then at least one term out of the terms encoding the clauses will be positive. □
ONE-IN-THREE 3SAT

- Has the same input as 3SAT.
- But asks whether there is a 0/1 assignment to the variables that in each clause satisfies exactly one literal.

\[
\begin{align*}
(x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3) \\
\end{align*}
\]

\[
\begin{align*}
x_1 = 1, \quad x_2 = 1, \quad x_3 = 0
\end{align*}
\] (satisfiable)

\[
\begin{align*}
(x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_3) \\
\end{align*}
\]

\[
\begin{align*}
\text{(unsatisfiable)}
\end{align*}
\]

- Reduction from 3SAT to ONE-IN-THREE 3SAT is on your homework.
**ONE-IN-THREE-3SAT $\rightarrow$ POLYPOS (degree 4)**

Almost the same construction as before, except ONE-IN-THREE-3SAT allows us to kill some terms and reduce the degree to 4. Nice!

Start with any instance of $3$SAT, such as:

$$
\Phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_3) \land (x_1 \lor x_3 \lor x_4)
$$

Construct $p$ as

$$
p(x) = \sum_{i=1}^{4} (x_i(1-x_i))^2 + \left[ (x_1+1-x_2+x_3-1)(x_1+1-x_2+x_3-2)(x_1+1-x_2+x_3-3) \right]^2
$$

**Moral:** Picking the tight problem for as the base problem of the reduction can make your life a lot simpler!

An aside: Testing convexity of quartics is also NP-hard! [AOPT13]
The knapsack problem

- **KNAPSACK**

- **Input:** A list of item values $p_1, \ldots, p_n$, a list of weights on the same items $w_1, \ldots, w_n$, two rational numbers $P, W$.

- **Question:** Can the thief steal a set of items of total value $\geq P$ that fit in his knapsack of total weight $W$?
The partition problem

**PARTITION**

**Input:** A list of positive integers $a_1, \ldots, a_n$.

**Question:** Can you split them into to bags such that the sum in one equals the sum in the other?

\[ \{5, 2, 1, 6, 3, 8, 5, 4, 1, 1, 10\} \quad \{5, 2, 1, 6, 3, 8, 5, 4, 1, 1, 10\} \]

- Note that the YES answer is easily verifiable.
- How would you efficiently verify a NO answer? (no one knows)
Testing polynomial positivity

**POLYPOS**

**Input:** A multivariate polynomial \( p(x) := p(x_1, \ldots, x_n) \) of degree four.

**Question:** Is there an \( x \in \mathbb{R}^n \) for which \( p(x) \leq 0 \)?

**Example:**
\[
p(x) = x_1^4 + 2x_1^2x_2^2 - 3x_1x_3 + 5x_2^4 + 6x_1^2x_2 - x_1x_2x_3 + 4x_3^4 + 100.
\]

A reduction from PARTITION to POLYPOS is on your homework.

Is there an easy certificate of the NO answer? (the answer is believed to be negative)

Is there an easy certificate of the YES answer? We don’t know; the obvious approach doesn’t work:
\[
p(x) = (x_1 - \lambda_1)^2 + (x_2 - \lambda_1)^2 + (x_3 - \lambda_2)^2 + \ldots + (x_n - \lambda_{n-1})^2
\]
\[
p(x) = 0 \implies \lambda_n = 2^{\frac{n}{2}}.
\]
But what about the first NP-complete problem?!!

- The Cook-Levin theorem.

- In a way a very deep theorem.

- At the same time almost a tautology.

- We argued in class how every problem in NP can be reduced to CIRCUIT SAT.
  - See Chapter 8 of [DPV].

CIRCUIT SAT $\rightarrow$ SAT $\rightarrow$ 3SAT (easy reductions)
The domino effect

- All NP-complete problems reduce to each other!
- If you solve one in polynomial time, you solve ALL in polynomial time!
Most people believe the answer is NO!
Philosophical reason: If a proof of the Goldbach conjecture were to fly from the sky, we could certainly efficiently verify it. But should this imply that we can find this proof efficiently? P=NP would imply the answer is yes.
Nevertheless, there are believers too...

- Over 100 wrong proofs have appeared so far (in both directions)! See http://www.win.tue.nl/~gwoegi/P-versus-NP.htm
Main messages...

- Computational complexity theory beautifully classifies many problems of optimization theory as easy or hard
  - At the most basic level, easy means “in P”, hard means “NP-hard.”
- The boundary between the two is very delicate:
  - MINCUT vs. MAXCUT, PENONPAPER vs. TSP, LP vs. IP, ...
- Important: When a problem is shown to be NP-hard, it doesn’t mean that we should give up all hope. NP-hard problems arise in applications all the time. There are good strategies for dealing with them.
  - Solving special cases exactly
  - Heuristics that work well in practice
  - Using convex optimization to find bounds and near optimal solutions
  - Approximation algorithms – suboptimal solutions with worst-case guarantees
- P=NP?
  - Maybe one of you guys will tell us one day.
Dealing with NP-hardness

1. **The SOS relaxation** for dealing with continuous NP-hard problems
   
   A very general and powerful framework with beautiful theory

2. **Branch & bound and cutting plane techniques** for dealing with discrete NP-hard problems
   
   Widely used in practice and very effective

3. **Approximation algorithms**
   
   Guaranteeing quality of suboptimal solutions in the worst case


- [AOPT13] NP-hardness of testing convexity: http://web.mit.edu/~a_a_a/Public/Publications/convexity_nphard.pdf