Integer Programming

Linear program = set of linear inequalities with rational coefficients.
- Feasible if there is a solution to all inequalities.

Example: $x \geq \frac{1}{2}$, $2x + y \leq 1$, $y \geq \frac{1}{2}$

Theorem: Linear Programming is in P

Theorem: Integer Programming is NP-Complete.

Proof:
Since we can easily check a proposed integer solution, the problem is in NP.

For showing that it is NP hard, reduce from 3-SAT.
For each clause like $(x_1 \lor \overline{x_2} \lor x_3)$, add inequality

$x_1 + (1 - x_2) + x_3 \geq 1.$

Furthermore, add inequalities $0 \leq x_i \leq 1$ to ensure that each variable has only two possible assignments: 0 (false) or 1 (true).
Note: Just because a problem is NP-complete does not mean every instance is hard.

- Example (Valiant '04)
  - Counting # of satisfying assignments to planar
    - #\text{Planar-Rt} is #\text{CNF} and not \text{NP-hard}
    - #\text{Planar-Rt} is \text{NP-hard}
  - #\text{Planar-Rt} is \text{NP-hard}

- Average-case complexity
  - Ajtai '96: a lattice problem in \text{NP} is hard in average provided another \text{NP} problem is hard on worst case.
Def: CLIQUE = \{<G, k> | G is an undirected graph with a k-clique, i.e., a set of k vertices any two of which are connected by an edge\}.

Ex:
```
   ∗—∗—∗
   ∗—∗—∗
   ∗—∗—∗
```

(a 3-clique (triangle) a 4-clique)

Theorem: CLIQUE is NP-complete.

Proof:
First, CLIQUE \in NP, here is a polytime verifier:
```
\[ \sqrt{G, k}, c : \]
\[ 1. \text{Test that } c \text{ is a set of } k \text{ nodes of } G \]
\[ 2. \text{Test that } G \text{ includes an edge between every pair of vertices in } c. \]
\[ 3. \text{If both pass, accept, else reject.} \]
```
(To get a nondeterministic Turing machine, just guess \( c \) first.)

To show that CLIQUE is NP-hard, we want to show that every NP language \( L \) reduces in polytime to CLIQUE. But since every \( L \) already reduces to 3-SAT (3-SAT is NP-hard) it is enough to reduce 3-SAT to CLIQUE.

Reduction: Input: a k-clause 3-SAT formula
\[ \phi = (a_1 \lor b_1 \lor c_1) \land (a_2 \lor b_2 \lor c_2) \land \cdots \land (a_k \lor b_k \lor c_k) \]
Output: \( <G, k> \), where \( G \) is given as follows

3 vertices for every clause
edges between all pairs of vertices for different clauses, except those for opposite literals, e.g., \( x_2 \) and \( \overline{x}_2 \)

Example: \( \phi = (x_1 \lor x_1 \lor x_2) \land (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor x_3 \lor x_4) \)
```
\[ G : \]
```
\[ \times \times \times \]
```

Claim: \( \phi \) is satisfiable iff \( G \) has a k-clique.
Claim: $\phi$ is satisfiable if and only if $G$ has a $k$-clique.

Proof: Write $\phi(x_1, \ldots, x_m) = C_1 \land C_2 \land \ldots \land C_k$, with $C_i = \exists i \exists V \bar{z} \exists x \exists z \bar{y}$.

$\Rightarrow$: Let $\phi$ be satisfiable by truth assignment $\tau$.

Choose one true $z_{ij}$ in each clause $C_i$ and add the corresponding vertex $v_{ij}$ to set $S$.

Clearly $|S| = k$.

Claim: $S$ is a clique.

Why? Consider any $v_{ij}, v_{kl} \in S$.

- $i \neq k$ since only one vertex per clause is in $S$.
- $z_{ij} \neq \bar{z}_{kl}$ as both $z_{ij}$ and $\bar{z}_{kl}$ are true.

$\Rightarrow$ edge $(v_{ij}, v_{kl})$ is in $G$.

$\Leftarrow$: Let $S$ be a clique of size $k$ in $G$.

Define truth assignment $\tau$ by $z_{ij} = \text{true}$ if $v_{ij} \in S$ (any unspecified variables may be set arbitrarily).

Claim 1: $\tau$ is a valid truth assignment.

Claim 2: $\phi$ is satisfied by $\tau$.

Proof of Claim 1:

$v_{ij}, v_{kl} \in S$ $\Rightarrow$ edge $(v_{ij}, v_{kl})$ in $G$.

$\Rightarrow z_{ij} \neq \bar{z}_{kl}$.

A variable and its negation cannot both be set true (no contradictions) $\square$.

Proof of Claim 2:

- No two vertices of $S$ are in the same clause.
- $|S| = k$.

$\Rightarrow$ Each clause has one variable in $S$.

$\Rightarrow$ Each clause has $\geq 1$ true literal.

$\Rightarrow$ Each clause is satisfied $\checkmark$.

DONE $\checkmark$.
Independent Set (I.S.)
- Graph \( G = (V, E) \)
- \( S \subseteq V \) is independent if for all \( u, v \in S \), the edge \( (u, v) \) is not in \( G \).

Instance: Graph \( G = (V, E) \), integer \( k \)
Problem: Does \( G \) have I.S. of size \( \geq k \) ?

Theorem: Independent Set is \( \text{NP} \)-complete.
Proof:
1) I.S. \( \leq_{\text{poly}} \text{NP} \) — Guess \( S \) and verify \( \checkmark \)
2) I.S. is \( \text{NP} \)-hard
   \( \text{CLIQUE} <_{\text{poly}} \text{I.S.} \)
   Reduction \( f \):
   - Input \( G = (V, E) \), integer \( k \)
   - Output \( \overline{G} \) complement of \( G \), integer \( k \)
   - \( \text{edge} \ (u, v) \) is in \( G \)
   - \( \text{edge} \ (u, v) \) is not in \( G \)
   - Clearly, Reduction \( f \) is poly-time computable.
   - Lemma: \( G \) has \( k \)-clique \( \iff \overline{G} \) has \( k \)-I.S.

Of course, following \( \text{CLIQUE} \), we can directly reduce \( 3\text{-SAT} <_{\text{poly}} \text{I.S.} \), \( \phi \),

\[ \phi = (x_1 \lor x_2 \lor \overline{x}_5) \land (x_2 \lor x_3 \lor \overline{x}_5) \land (\overline{x}_1 \lor x_4 \lor x_5) \land (\overline{x}_1 \lor \overline{x}_5 \lor x_5) \]

Alternatively, observe that the complement of an independent set is a vertex cover, so we could have used reductions

\( 3\text{-SAT} <_{\text{poly}} \text{VERTEX-COVER} <_{\text{poly}} \text{I.S.} <_{\text{poly}} \text{CLIQUE} \).
Graph coloring problems

**Def:** A coloring of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color.

**Theorem:** Let \[ \text{SCOLOR} = \{<G> \mid \text{the nodes of } G \text{ can be colored with three colors so no two nodes joined by an edge have the same color}\} \]

Then \text{SCOLOR} is an NP-complete problem.

**Proof sketch:** \text{SCOLOR} \in NP clearly.

For NP-hardness, reduce from 3SAT.

Use the gadgets

![Gadgets](image)

**Theorem (Blum & Karger '97):** Assuming G is a 3-colorable graph, there is a polynomial-time algorithm that colors it using at most \( n \ln n \cdot (\log n)^{O(1)} \) colors.

**Theorem (Khanna, Linial, Safra '93):** If P \neq NP, there is no poly-time algorithm that colors 3-colorable graphs using 4 colors.

--huge gap!
**Subset Sum**  
Instance: Collection of numbers \(x_1, \ldots, x_k\) (allowing repetition)  
Target number \(t\)  
Problem: Is there a subcollection that adds to \(t\)?

\[
\text{SUBSET-SUM} = \left\{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\} \text{ and for some } \{y_1, \ldots, y_k\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \sum y_i = t \right\}
\]

\[
\text{Ex.}: \langle \{4, 11, 16, 21, 27, 3, 25\} \rangle \in \text{SUBSET-SUM} \quad (4 + 21 = 25)
\]

**SUBSET-SUM \in \text{NP}** clearly. It is **not obvious** that the complement **SUBSET-SUM** is in \text{NP}. **Def:** \text{co-NP} = class of languages whose complements are in \text{NP}.

**Theorem:** **SUBSET-SUM** is \text{NP}-complete.

**Proof:** By a reduction from 3-SAT  
**Idea:**  
1. Represent variable \(x_i\) by two numbers \(y_i\) and \(z_i\).  
2. Show that exactly one must be in any subcollection summing to the target — encoding \(x_i's\) truth value  
3. Add constraints also for every clause

Let \(\phi\) be a boolean formula with variables \(x_1, \ldots, x_k\), clauses \(c_1, \ldots, c_k\),  
\[
eq \quad (x_1 \lor x_2 \lor x_3) \land (x_2 \lor x_3 \lor v \ldots) \land \ldots \land (x_3 \lor v \ldots v)
\]

Becomes

<table>
<thead>
<tr>
<th>(y_1)</th>
<th>(y_2)</th>
<th>(y_3)</th>
<th>(y_4)</th>
<th>(y_5)</th>
<th>(y_6)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(c_1)</th>
<th>(c_2)</th>
<th>(c_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
<td>(x_4)</td>
<td>(x_5)</td>
<td>(x_6)</td>
<td>(c_1)</td>
<td>(c_2)</td>
<td>(c_3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table size is \(\sim (k+1)^k\), so reduction is computable in poly-time

**Claim:** \(\phi\) is satisfiable if and only if a subset of \(S\) sums to \(t\).

(Observe: Since there are at most \(5\) ones in any column, "carries" are impossible (base 10)).
Time Hierarchy Theorem

Theorem [Simple case]:

There exists a language $L$ that is decidable in time $O(n^{1.5})$ but not in time $O(n)$.

Proof: Let $D$ be the following Turing machine:

1. **Input**: $x \in \{0,1\}^n$
2. Run the universal Turing machine $U$ to simulate $M_x$ (Turing machine encoded by $x$) on input $x = \langle M_x \rangle$ for $n^{1.5}$ steps.
3. If $M_x$ outputs an answer, then output the opposite! Otherwise reject.

Let $L = L(D)$.

- $L$ is decidable in time $O(n^{1.5})$.
- **Claim**: $L$ is not decidable in time $O(n)$
  
  **Proof**: Assume otherwise.

Let $M$ be a Turing machine that decides $L$ (i.e. $M(x) = D(x)$ for all $x \in \{0,1\}^n$) and on inputs of length $n \gg n_0$, $M$ runs in time $\leq c \cdot n$.

Let $\langle M \rangle$ be an encoding of $M$ of length $n \gg n_0$, large enough so $c \cdot n \leq n^{1.5}$.

(Such an encoding exists because every Turing machine is represented by infinitely many strings)

Then $\langle M \rangle \in L \iff D$ accepts $M$

$\iff M$ rejects $\langle M \rangle$ in $\leq n^{1.4}$ steps

$\iff M$ rejects $\langle M \rangle$ (since time $c \cdot n \leq n^{1.5}$)

$\iff \langle M \rangle \notin L$ — contradiction! □

More generally,

Theorem: If $f, g$ are time-constructible functions, $f(n) \log f(n) = o(g(n))$,

Then $\text{TIME}(f(n)) \neq \text{TIME}(g(n))$

("Time constructible" means there is a Turing machine that on input $1^n$

writes $1^{f(n)}$ on its tape in $O(f(n))$ time)