More on SAT

Lecture 23
April 29, 2021
Part I

Circuit SAT
Circuits

**Definition**

A circuit is a directed *acyclic* graph with

1. **Input** vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled \( \lor, \land \) or \( \neg \).
3. Single node **output** vertex with no outgoing edges.

Inputs: 1, ?, ?, 0, ?
Circuits

**Definition**

A circuit is a directed *acyclic* graph with

1. Input vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.
Definition

A circuit is a directed *acyclic* graph with

1. Input vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled $\lor$, $\land$ or $\lnot$.
3. Single node output vertex with no outgoing edges.
Definition (Circuit Satisfaction (CSAT).)
Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?
**CSAT: Circuit Satisfaction**

**Definition (Circuit Satisfaction (CSAT).)**

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

**Claim**

**CSAT** is in **NP**.

1. **Certificate:** Assignment to input variables.
2. **Certifier:** Evaluate the value of each gate in a topological sort of **DAG** and check the output gate value.
Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas.
Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

However they are equivalent in terms of polynomial-time solvability.

**Theorem**

\[
\text{SAT} \leq_p \text{3SAT} \leq_p \text{CSAT}.
\]

**Theorem**

\[
\text{CSAT} \leq_p \text{SAT} \leq_p \text{3SAT}.
\]
Converting a \textbf{CNF} formula into a Circuit

Given 3\textbf{CNF} formula $\varphi$ with $n$ variables and $m$ clauses, create a Circuit $C$.

- Inputs to $C$ are the $n$ boolean variables $x_1, x_2, \ldots, x_n$
- Use NOT gate to generate literal $\neg x_i$ for each variable $x_i$
- For each clause $(\ell_1 \lor \ell_2 \lor \ell_3)$ use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output
Example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4) \]
Converting a circuit into a CNF formula

Label the nodes

(A) Input circuit

(B) Label the nodes.
Converting a circuit into a **CNF formula**

Introduce a variable for each node

(B) Label the nodes.

(C) Introduce var for each node.
Converting a circuit into a CNF formula

Write a sub-formula for each variable that is true if the var is computed correctly.

(C) Introduce var for each node.

(D) Write a sub-formula for each variable that is true if the var is computed correctly.

\( x_k \) (Demand a sat’ assignment!)
\( x_k = x_i \land x_j \)
\( x_j = x_g \land x_h \)
\( x_i = \neg x_f \)
\( x_h = x_d \lor x_e \)
\( x_g = x_b \lor x_c \)
\( x_f = x_a \land x_b \)
\( x_d = 0 \)
\( x_a = 1 \)
Converting a circuit into a **CNF formula**

Convert each sub-formula to an equivalent **CNF formula**

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$x_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_k = x_i \land x_j$</td>
<td>$(\neg x_k \lor x_i) \land (\neg x_k \lor x_j) \land (x_k \lor \neg x_i \lor \neg x_j)$</td>
</tr>
<tr>
<td>$x_j = x_g \land x_h$</td>
<td>$(\neg x_j \lor x_g) \land (\neg x_j \lor x_h) \land (x_j \lor \neg x_g \lor \neg x_h)$</td>
</tr>
<tr>
<td>$x_i = \neg x_f$</td>
<td>$(x_i \lor x_f) \land (\neg x_i \lor \neg x_f)$</td>
</tr>
<tr>
<td>$x_h = x_d \lor x_e$</td>
<td>$(x_h \lor \neg x_d) \land (x_h \lor \neg x_e) \land (\neg x_h \lor x_d \lor x_e)$</td>
</tr>
<tr>
<td>$x_g = x_b \lor x_c$</td>
<td>$(x_g \lor \neg x_b) \land (x_g \lor \neg x_c) \land (\neg x_g \lor x_b \lor x_c)$</td>
</tr>
<tr>
<td>$x_f = x_a \land x_b$</td>
<td>$(\neg x_f \lor x_a) \land (\neg x_f \lor x_b) \land (x_f \lor \neg x_a \lor \neg x_b)$</td>
</tr>
<tr>
<td>$x_d = 0$</td>
<td>$\neg x_d$</td>
</tr>
<tr>
<td>$x_a = 1$</td>
<td>$x_a$</td>
</tr>
</tbody>
</table>
Converting a circuit into a CNF formula

Take the conjunction of all the CNF sub-formulas

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.
Reduction: \( \text{CSAT} \leq_p \text{SAT} \)

1. For each gate (vertex) \( v \) in the circuit, create a variable \( x_v \)

2. Case \( \neg \): \( v \) is labeled \( \neg \) and has one incoming edge from \( u \) (so \( x_v = \neg x_u \)). In \( \text{SAT} \) formula generate, add clauses \( (x_u \lor x_v) \), \( (\neg x_u \lor \neg x_v) \). Observe that

\[
x_v = \neg x_u \text{ is true } \iff (x_u \lor x_v) \land (\neg x_u \lor \neg x_v) \text{ both true.}
\]
Case ∨: So $x_v = x_u \lor x_w$. In SAT formula generated, add clauses $(x_v \lor \neg x_u)$, $(x_v \lor \neg x_w)$, and $(\neg x_v \lor x_u \lor x_w)$. Again, observe that

$$(x_v = x_u \lor x_w) \text{ is true } \iff (x_v \lor \neg x_u), (x_v \lor \neg x_w), (\neg x_v \lor x_u \lor x_w) \text{ all true.}$$
Reduction: $\text{CSAT} \leq_P \text{SAT}$

Continued...

1. Case $\land$: So $x_v = x_u \land x_w$. In SAT formula generated, add clauses $(\neg x_v \lor x_u)$, $(\neg x_v \lor x_w)$, and $(x_v \lor \neg x_u \lor \neg x_w)$. Again observe that

$$x_v = x_u \land x_w \text{ is true } \iff (\neg x_v \lor x_u), \quad (\neg x_v \lor x_w), \quad (x_v \lor \neg x_u \lor \neg x_w) \text{ all true.}$$
Reduction: **CSAT \( \leq_P \) SAT**

Continued...

1. If \( v \) is an input gate with a fixed value then we do the following. If \( x_v = 1 \) add clause \( x_v \). If \( x_v = 0 \) add clause \( \neg x_v \).

2. Add the clause \( x_v \) where \( v \) is the variable for the output gate.
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$

1. Find values of all gates in $C$ under $a$
2. Give value of gate $v$ to variable $x_v$; call this assignment $a'$
3. $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$

1. Let $a'$ be the restriction of $a$ to only the input variables
2. Value of gate $v$ under $a'$ is the same as value of $x_v$ in $a$
3. Thus, $a'$ satisfies $C$
Part II

SAT reduces to 3-SAT
How SAT is different from 3SAT?

In SAT clauses might have arbitrary length: 1, 2, 3, ..., variables:

\[(x \lor y \lor z \lor w \lor u) \land (\neg x \lor \neg y \lor \neg z \lor w \lor u) \land (\neg x)\]

In 3SAT every clause must have exactly 3 different literals.
SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

In SAT clauses might have arbitrary length: $1, 2, 3, \ldots$ variables:

$$
\left( x \lor y \lor z \lor w \lor u \right) \land \left( \neg x \lor \neg y \lor \neg z \lor w \lor u \right) \land \left( \neg x \right)
$$

In 3SAT every clause must have exactly 3 different literals.

To reduce from an instance of SAT to an instance of 3SAT, we must make all clauses to have exactly 3 variables...

Basic idea

1. Pad short clauses so they have 3 literals.
2. Break long clauses into shorter clauses.
3. Repeat the above till we have a 3CNF.
3SAT $\leq_P$ SAT

1. 3SAT $\leq_P$ SAT.

2. Because...
   A 3SAT instance is also an instance of SAT.
Claim

SAT \leq_P 3SAT.

Given \( \phi \) a SAT formula we create a 3SAT formula \( \phi' \) such that

1. \( \phi \) is satisfiable iff \( \phi' \) is satisfiable.
2. \( \phi' \) can be constructed from \( \phi \) in time polynomial in \( |\phi| \).

Idea: if a clause of \( \phi \) is not of length 3, replace it with several clauses of length exactly 3.
Claim

\[ \text{SAT} \leq_P \text{3SAT} \]

Given \( \varphi \) a \textbf{SAT} formula we create a \textbf{3SAT} formula \( \varphi' \) such that

1. \( \varphi \) is satisfiable iff \( \varphi' \) is satisfiable.
2. \( \varphi' \) can be constructed from \( \varphi \) in time polynomial in \( |\varphi| \).
Claim

\( \text{SAT} \leq_p \text{3SAT} \).

Given \( \phi \) a \( \text{SAT} \) formula we create a \( \text{3SAT} \) formula \( \phi' \) such that

1. \( \phi \) is satisfiable iff \( \phi' \) is satisfiable.
2. \( \phi' \) can be constructed from \( \phi \) in time polynomial in \( |\phi| \).

Idea: if a clause of \( \phi \) is not of length 3, replace it with several clauses of length exactly 3.
SAT $\leq_P$ 3SAT
A clause with two literals

Reduction Ideas: clause with 2 literals

1. Case clause with 2 literals: Let $c = \ell_1 \lor \ell_2$. Let $u$ be a new variable. Consider

$$c' = (\ell_1 \lor \ell_2 \lor u) \land (\ell_1 \lor \ell_2 \lor \neg u).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
SAT $\leq_P$ 3SAT

A clause with a single literal

Reduction Ideas: clause with 1 literal

1. Case clause with one literal: Let $c$ be a clause with a single literal (i.e., $c = \ell$). Let $u, v$ be new variables. Consider

$$c' = (\ell \lor u \lor v) \land (\ell \lor u \lor \neg v)$$

$$\land (\ell \lor \neg u \lor v) \land (\ell \lor \neg u \lor \neg v).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
SAT $\leq_P$ 3SAT

A clause with more than 3 literals

Reduction Ideas: clause with more than 3 literals

1. Case clause with five literals: Let $c = \ell_1 \lor \ell_2 \lor \ell_3 \lor \ell_4 \lor \ell_5$. Let $u$ be a new variable. Consider

$$c' = (\ell_1 \lor \ell_2 \lor \ell_3 \lor u) \land (\ell_4 \lor \ell_5 \lor \neg u).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
SAT $\leq_P$ 3SAT

A clause with more than 3 literals

Reduction Ideas: clause with more than 3 literals

1. Case clause with $k > 3$ literals: Let $c = l_1 \lor l_2 \lor \ldots \lor l_k$. Let $u$ be a new variable. Consider

$$c' = (l_1 \lor l_2 \ldots l_{k-2} \lor u) \land (l_{k-1} \lor l_k \lor \neg u).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
Breaking a clause

Lemma

For any boolean formulas $X$ and $Y$ and $z$ a new boolean variable. Then

$$X \lor Y \text{ is satisfiable}$$

if and only if, $z$ can be assigned a value such that

$$(X \lor z) \land (Y \lor \neg z) \text{ is satisfiable}$$

(with the same assignment to the variables appearing in $X$ and $Y$).
SAT \lesssim_P \text{3SAT} (\text{contd})

Clauses with more than 3 literals

Let \( c = \ell_1 \lor \cdots \lor \ell_k \). Let \( u_1, \ldots, u_{k-3} \) be new variables. Consider

\[
c' = (\ell_1 \lor \ell_2 \lor u_1) \land (\ell_3 \lor \neg u_1 \lor u_2) \land (\ell_4 \lor \neg u_2 \lor u_3) \land \cdots \land (\ell_{k-2} \lor \neg u_{k-4} \lor u_{k-3}) \land (\ell_{k-1} \lor \ell_k \lor \neg u_{k-3}).
\]

Claim

\( \varphi = \psi \land c \) is satisfiable iff \( \varphi' = \psi \land c' \) is satisfiable.

Another way to see it — reduce size of clause by one:

\[
c' = (\ell_1 \lor \ell_2 \cdots \lor \ell_{k-2} \lor u_{k-3}) \land (\ell_{k-1} \lor \ell_k \lor \neg u_{k-3}).
\]
An Example

Example

\( \varphi = \left( \neg x_1 \lor \neg x_4 \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor x_4 \lor x_1 \right) \land \left( x_1 \right) \).

Equivalent form:

\( \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \).
An Example

Example

\[ \varphi = \left( \neg x_1 \lor \neg x_4 \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor x_4 \lor x_1 \right) \land \left( x_1 \right). \]

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \]
An Example

Example

\[ \varphi = \left( \neg x_1 \lor \neg x_4 \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor x_4 \lor x_1 \right) \land \left( x_1 \right). \]

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor y_1 \right) \land \left( x_4 \lor x_1 \lor \neg y_1 \right). \]
An Example

Example

\[ \varphi = \left( \neg x_1 \lor \neg x_4 \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor x_4 \lor x_1 \right) \land \left( x_1 \right) . \]

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor y_1 \right) \land \left( x_4 \lor x_1 \lor \neg y_1 \right) \land \left( x_1 \lor u \lor \neg v \right) \land \left( x_1 \lor \neg u \lor \neg v \right) . \]
Overall Reduction Algorithm
Reduction from SAT to 3SAT

ReduceSATTo3SAT(\(\varphi\)):

// \(\varphi\): CNF formula.
for each clause \(c\) of \(\varphi\) do
    if \(c\) does not have exactly 3 literals then
        construct \(c'\) as before
    else
        \(c' = c\)
\(\psi\) is conjunction of all \(c'\) constructed in loop
return Solver3SAT(\(\psi\))

Correctness (informal)
\(\varphi\) is satisfiable iff \(\psi\) is satisfiable because for each clause \(c\), the new 3CNF formula \(c'\) is logically equivalent to \(c\).
Part III

Reducing Problems to SAT and Circuit SAT
Power of SAT and CSAT

**SAT** and **CSAT** are meta-problems

Allow us to express/model problem using constraints. In essence they allow programming with constraints of certain restricted type.

**Goal:** examples to drive home the point
Reduce Directed Hamilton Path to SAT

Given directed graph $G = (V, E)$, does it have a Hamilton path?

Given $G$ obtain CNF formula $\varphi_G$ such that $G$ has a Hamilton Path iff $\varphi_G$ is satisfiable
Reduce Directed Hamilton Path to SAT

Given directed graph $G = (V, E)$, does it have a Hamilton path?

Given $G$ obtain CNF formula $\varphi_G$ such that $G$ has a Hamilton Path iff $\varphi_G$ is satisfiable

**Alternative view:** Program/express using constraints

- What are variables?
- What are the constraints?
Given directed graph $G = (V, E)$, does it have a Hamilton path?

Given $G$ obtain CNF formula $\varphi_G$ such that $G$ has a Hamilton Path iff $\varphi_G$ is satisfiable

**Alternative view:** Program/express using constraints

- What are variables?
- What are the constraints?

**One approach:** $G$ has a Hamilton path iff there is a permutation of the $n$ vertices such that for each $i$ there is an edge from vertex in position $i$ to vertex in position $(i + 1)$

How do we express permutations?
Define variable $x(u, i)$ if vertex $u$ in position $i$ in the permutation. Total of $n^2$ variables where $n = |V|$.

Constraints?

- For each $u$, exactly one of $x(u, 1), x(u, 2), \ldots, x(u, n)$ should be true.
Define variable $x(u, i)$ if vertex $u$ in position $i$ in the permutation. Total of $n^2$ variables where $n = |V|$.

Constraints?

- For each $u$, exactly one of $x(u, 1), x(u, 2), \ldots, x(u, n)$ should be true
  - $\bigvee_{i=1}^{n} x(u, i)$ to ensure that $x(u, i)$ is 1 for at least one $i$
  - For $i \neq j$ we add constraint $\neg x(u, i) \lor \neg x(u, j)$ to ensure that we cannot choose both to be 1 for any pair.
- For each $u$ we have a total of $(1 + n(n - 1)/2)$ constraints. Total of $n(1 + n(n - 1)/2)$ over all vertices.
- $x(u, i)$ and $x(v, i + 1)$ implies edge $(u, v)$ in $E(G)$
Define variable $x(u, i)$ if vertex $u$ in position $i$ in the permutation. Total of $n^2$ variables where $n = |V|$.

Constraints?

- For each $u$, exactly one of $x(u, 1), x(u, 2), \ldots, x(u, n)$ should be true.
  - $\bigvee_{i=1}^n x(u, i)$ to ensure that $x(u, i)$ is 1 for at least one $i$
  - For $i \neq j$ we add constraint $\neg x(u, i) \lor \neg x(u, j)$ to ensure that we cannot choose both to be 1 for any pair.

- For each $u$ we have a total of $(1 + n(n - 1)/2)$ constraints. Total of $n(1 + n(n - 1)/2)$ over all vertices.

- $x(u, i)$ and $x(v, i + 1)$ implies edge $(u, v)$ in $E(G)$
  - $(x(u, i) \land x(v, i + 1)) \Rightarrow z(u, v)$ where $z(u, v)$ is 1 if $(u, v) \in E$ otherwise 0 ($z(u, v)$ is a constant, not a variable but to help notation). Convert implication constraint to CNF.
Vertex Cover to CSAT

Given graph $G = (V, E)$ and integer $k$, does $G$ have a vertex cover of size at most $k$?

Recall $S \subseteq V$ is a vertex cover if each edge $(u, v)$ is covered by $S$, that means $u \in S$ or $v \in S$.

How do we reduce to CSAT/SAT? What are the variables?
Vertex Cover to CSAT

Given graph $G = (V, E)$ and integer $k$, does $G$ have a vertex cover of size at most $k$?

Recall $S \subseteq V$ is a vertex cover if each edge $\langle u, v \rangle$ is covered by $S$, that means $u \in S$ or $v \in S$.

How do we reduce to CSAT/SAT? What are the variables? $x_u, u \in V$ to indicate whether we choose $u$.
Vertex Cover to CSAT

Given graph $G = (V, E)$ and integer $k$, does $G$ have a vertex cover of size at most $k$?

Recall $S \subseteq V$ is a vertex cover if each edge $(u, v)$ is covered by $S$, that means $u \in S$ or $v \in S$.

How do we reduce to CSAT/SAT? What are the variables? $x_u, u \in V$ to indicate whether we choose $u$

Constraints?
Vertex Cover to CSAT

Given graph $G = (V, E)$ and integer $k$, does $G$ have a vertex cover of size at most $k$?

Recall $S \subseteq V$ is a vertex cover if each edge $(u, v)$ is covered by $S$, that means $u \in S$ or $v \in S$.

How do we reduce to CSAT/SAT? What are the variables? $x_u, u \in V$ to indicate whether we choose $u$

Constraints?

- For each edge $(u, v) \in E$ a constraint $(x_u \lor x_v)$. Total of $|E|$ constraints.
**Vertex Cover to CSAT**

Given graph $G = (V, E)$ and integer $k$, does $G$ have a vertex cover of size at most $k$?

Recall $S \subseteq V$ is a vertex cover if each edge $(u, v)$ is covered by $S$, that means $u \in S$ or $v \in S$.

How do we reduce to CSAT/SAT? What are the variables? $x_u$, $u \in V$ to indicate whether we choose $u$.

Constraints?

- For each edge $(u, v) \in E$ a constraint $(x_u \lor x_v)$. Total of $|E|$ constraints.
- $\sum_{u \in V} x_u \leq k$. Not a boolean constraint! How?
Vertex Cover to CSAT

Expressing $\sum_{u \in V} x_u \leq k$ as a circuit.

- Given inputs $x_u, u \in V$ can create an addition circuit that outputs the sum $\sum_u x_u$ as a $\lceil \log n \rceil$ bit binary number.
- Given two $r$-bit binary inputs $y_1, y_2, \ldots, y_r$ and $z_1, z_2, \ldots, z_r$ one can develop a boolean circuit to compare which one is greater.
- Hence circuit to do $\sum_u x_u$ and compare output to input integer $k$ written in binary.
Expressing $\sum_{u \in V} x_u \leq k$ as a circuit.

- Given inputs $x_u$, $u \in V$ can create an addition circuit that outputs the sum $\sum_u x_u$ as a $\lceil \log n \rceil$ bit binary number.
- Given two $r$-bit binary inputs $y_1, y_2, \ldots, y_r$ and $z_1, z_2, \ldots, z_r$ one can develop a boolean circuit to compare which one is greater.
- Hence circuit to do $\sum_u x_u$ and compare output to input integer $k$ written in binary.

Combine with the constraints to cover edges to obtain a CSAT instance with input variables $x_u$, $u \in V$. 

Vertex Cover to CSAT
Cook-Levin Theorem

Theorem (Cook-Levin)

**SAT is NP-Complete.**

How did they prove it? And why SAT or CSAT?

Proof is in retrospect simple.

- Fix any non-deterministic TM $M$ and string $w$.
- Does $M$ accept $w$ in $p(|w|)$ steps where $p()$ is some fixed polynomial?
- Can express computation of $M$ on $w$ using a polynomial sized circuit (or CNF formula) due to expressive power of constraints and local computation of TMs.
- Thus, can reduce an arbitrary NP problem (since it corresponds to some non-deterministic poly-time TM $M$) to SAT.
**SAT, CSAT** are boolean constraint satisfaction problems.

**Other frameworks:** constraints involving linear inequalities, convex functions, polynomials etc

**Useful to know:** Integer Linear Programming (ILP), Linear Programming (LP), Mixed Integer Linear Programming (MIP), Convex Programming

Commercial packages available. ILP, MIP are NP-Hard but many small to medium problems can be solved in practice. Powerful and expressive constraint involving numbers, not just booleans.
Linear Programming

Problem

Real variables $x_1, x_2, \ldots, x_n$. Solve

maximize/minimize $\sum_{j=1}^{n} c_j x_j$

subject to

$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for $i = 1 \ldots p$

$\sum_{j=1}^{n} a_{ij} x_j = b_i$ for $i = p + 1 \ldots q$

$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ for $i = q + 1 \ldots m$

Input is matrix $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, column vector $b = (b_i) \in \mathbb{R}^{m}$, and row vector $c = (c_j) \in \mathbb{R}^{n}$

Constraints are linear equations and inequalities. Objective is a linear function
Integer Linear Programming

**Problem**

**Integer** variables $x_1, x_2, \ldots, x_n$. Solve

maximize/minimize $\sum_{j=1}^{n} c_j x_j$

subject to

$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for $i = 1 \ldots p$

$\sum_{j=1}^{n} a_{ij} x_j = b_i$ for $i = p + 1 \ldots q$

$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ for $i = q + 1 \ldots n$

$x_i \in \mathbb{Z}$ for $i = 1$ to $d$

Input is matrix $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, column vector $b = (b_i) \in \mathbb{R}^{m}$, and row vector $c = (c_j) \in \mathbb{R}^{n}$

Constraints are linear equations and inequalities. Objective is a linear function but variables need to take integer values.
Convex Programming

Problem

Real variables \( x_1, x_2, \ldots, x_n. \ x \in \mathbb{R}^n \) Solve

minimize \( f(x) \)
subject to \( g_i(x) \leq b_i \) for \( i = 1 \ldots m \)

\( f, g_1, g_2, \ldots, g_m \) are convex functions
Mathematical Programming

- LP is a special case of Convex Programming
- LP can be solved in polynomial time
- Convex programs can be solved arbitrarily well in polynomial time (exact solution is tricky because of irrational solutions)
- ILP and MIP are NP-Hard (decision versions are NP-Complete).
Mathematical Programming

- LP is a special case of Convex Programming
- LP can be solved in polynomial time
- Convex programs can be solved arbitrarily well in polynomial time (exact solution is tricky because of irrational solutions)
- ILP and MIP are NP-Hard (decision versions are NP-Complete).

Why is convex programming solvable?

- For convex programs, local optimum is a global optimum!
- Local optimum can be found by local search! Gradient descent!
- Even for non-convex programs
- Gradient descent doesn’t give a poly-time algorithm (gives a pseudo-polytime algorithm) but shows why efficiency is possible.
Interplay of Discrete and Continuous Optimization

Both are fundamental and important and interplay has lot of impact!

- Machine learning: (deep) learning uses continuous optimization to train neural networks for classification and other discrete tasks
- Combinatorial optimization: use LP/SDP and other convex programming methods to solve combinatorial problems
- Scientific and numerical computing
- Statistics
- ...