More on SAT

Lecture 23

April 29, 2021

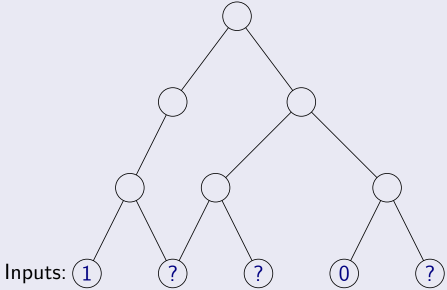
Part I

Circuit SAT

Circuits

Definition

A circuit is a directed *acyclic* graph with

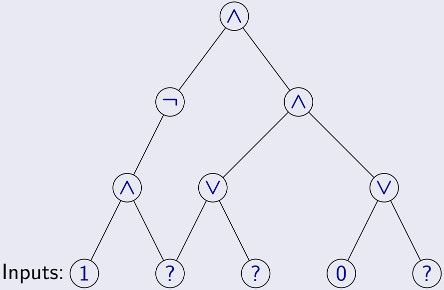


1. Input vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

Circuits

Definition

A circuit is a directed *acyclic* graph with

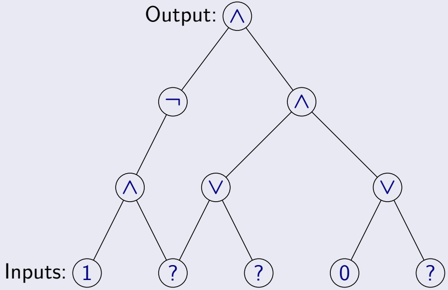


1. Input vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

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.

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?

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

SAT $\leq_P$ **3SAT** $\leq_P$ **CSAT**.

Theorem

CSAT $\leq_P$ **SAT** $\leq_P$ **3SAT**.

Converting a CNF formula into a Circuit

Given $3\mathrm{CNF}$ formulat φ with n variables and m clauses, create a Circuit C.

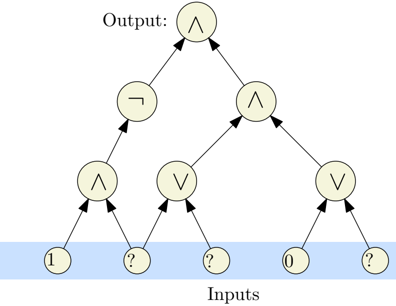
- Inputs to C are the n boolean variables $x_1, x_2, \ldots, x_n$
- Use NOT gate to generate literal $\neg x_i$ for each variable x_i
- For each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output

Example

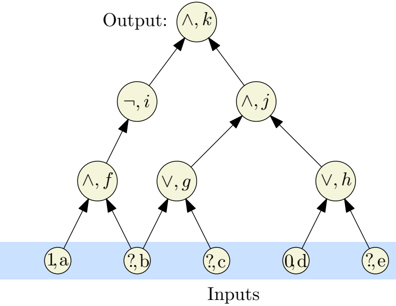
$$\varphi = \left(x_1 \lor \lor x_3 \lor x_4\right) \land \left(x_1 \lor \neg x_2 \lor \neg x_3\right) \land \left(\neg x_2 \lor \neg x_3 \lor x_4\right)$$

Converting a circuit into a CNF formula

Label the nodes



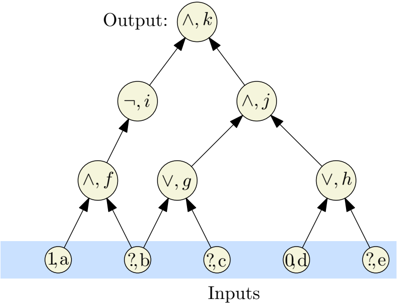
(A) Input circuit



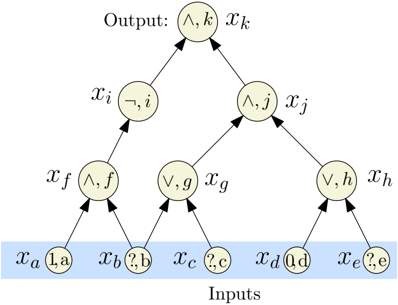
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node



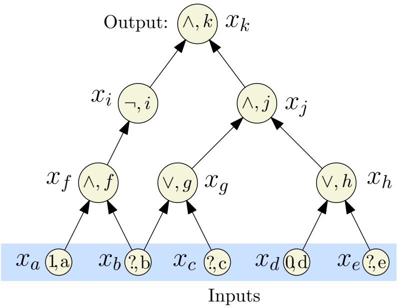
(B) Label the nodes.



(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.

$\boldsymbol{x_k}$ (Demand a sat' assignment!)

$\boldsymbol{x_k = x_i \wedge x_j}$

$\boldsymbol{x_j = x_g \wedge x_h}$

$\boldsymbol{x_i = \neg x_f}$

$\boldsymbol{x_h = x_d \vee x_e}$

$\boldsymbol{x_g = x_b \vee x_c}$

$\boldsymbol{x_f = x_a \wedge x_b}$

$\boldsymbol{x_d = 0}$

$\boldsymbol{x_a = 1}$

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

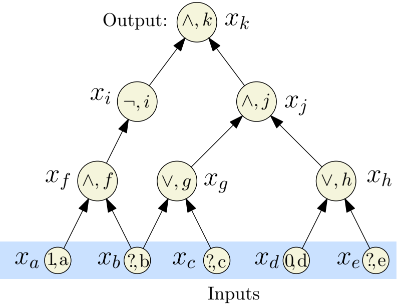
Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula

x_k	x_k
$x_k = x_i \wedge x_j$	$(\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \wedge (x_k \vee \neg x_i \vee \neg x_j)$
$x_j = x_g \wedge x_h$	$(\neg x_j \vee x_g) \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h)$
$x_i = \neg x_f$	$(x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f)$
$x_h = x_d \vee x_e$	$(x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \wedge (\neg x_h \vee x_d \vee x_e)$
$x_g = x_b \vee x_c$	$(x_g \vee \neg x_b) \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c)$
$x_f = x_a \wedge x_b$	$(\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \wedge (x_f \vee \neg x_a \vee \neg x_b)$
$x_d = 0$	$\neg x_d$
$x_a = 1$	x_a

Converting a circuit into a CNF formula

Take the conjunction of all the CNF sub-formulas



$$
\begin{aligned}
& x_k \wedge (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \\
& \wedge (x_k \vee \neg x_i \vee \neg x_j) \wedge (\neg x_j \vee x_g) \\
& \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h) \\
& \wedge (x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f) \\
& \wedge (x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \\
& \wedge (\neg x_h \vee x_d \vee x_e) \wedge (x_g \vee \neg x_b) \\
& \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c) \\
& \wedge (\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \\
& \wedge (x_f \vee \neg x_a \vee \neg x_b) \wedge (\neg x_d) \wedge x_a
\end{aligned}
$$

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.

Reduction: CSAT $\leq_P$ 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 **SAT** formula generate, add clauses $(x_u \vee x_v)$, $(\neg x_u \vee \neg x_v)$. Observe that

$$x_v = \neg x_u \text{ is true} \quad \iff \quad \begin{matrix} (x_u \vee x_v) \\ (\neg x_u \vee \neg x_v) \end{matrix} \text{ both true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

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

$$\Big(x_v = x_u \vee x_w\Big) \text{ is true} \quad \iff \quad \begin{array}{l} (x_v \vee \neg x_u), \\ (x_v \vee \neg x_w), \\ (\neg x_v \vee x_u \vee x_w) \end{array} \quad \text{all true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

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

$$x_v = x_u \wedge x_w \text{ is true} \iff \begin{array}{l}(\neg x_v \vee x_u),\\ (\neg x_v \vee x_w),\\ (x_v \vee \neg x_u \vee \neg x_w)\end{array} \quad \text{all true.}$$

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 φ_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 φ_C (exercise)

$\Leftarrow$ Consider a satisfying assignment a for φ_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

SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

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

$$\left(x \vee y \vee z \vee w \vee u\right) \wedge \left(\neg x \vee \neg y \vee \neg z \vee w \vee u\right) \wedge \left(\neg x\right)$$

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:

$$\left(x \vee y \vee z \vee w \vee u\right) \wedge \left(\neg x \vee \neg y \vee \neg z \vee w \vee u\right) \wedge \left(\neg x\right)$$

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 $3\mathrm{CNF}$.

3SAT $\leq_P$ SAT

1. **3SAT** $\leq_P$ **SAT**.
2. Because...
   A **3SAT** instance is also an instance of **SAT**.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ 3SAT.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ **3SAT**.

Given φ a **SAT** formula we create a **3SAT** formula φ' such that

1. φ is satisfiable iff φ' is satisfiable.
2. φ' can be constructed from φ in time polynomial in $|\varphi|$.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ **3SAT**.

Given φ a **SAT** formula we create a **3SAT** formula φ' such that

1. φ is satisfiable iff φ' is satisfiable.
2. φ' can be constructed from φ in time polynomial in $|\varphi|$.

Idea: if a clause of φ is not of length 3, replace it with several clauses of length exactly 3.

Reduction Ideas: clause with 2 literals

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

$$c' = \Big(\ell_1 \vee \ell_2 \vee u\Big) \wedge \Big(\ell_1 \vee \ell_2 \vee \neg u\Big).$$

2. Suppose $\varphi = \psi \wedge c$. Then $\varphi' = \psi \wedge c'$ is satisfiable iff φ 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

$$c' = \left(\ell \vee u \vee v\right) \wedge \left(\ell \vee u \vee \neg v\right) \wedge \left(\ell \vee \neg u \vee v\right) \wedge \left(\ell \vee \neg u \vee \neg v\right).$$

2. Suppose $\varphi = \psi \wedge c$. Then $\varphi' = \psi \wedge c'$ is satisfiable iff φ 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 $c = \ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4 \vee \ell_5$. Let u be a new variable. Consider

$$c' = \Big(\ell_1 \vee \ell_2 \vee \ell_3 \vee u\Big) \wedge \Big(\ell_4 \vee \ell_5 \vee \neg u\Big).$$

2. Suppose $\varphi = \psi \wedge c$. Then $\varphi' = \psi \wedge c'$ is satisfiable iff φ is satisfiable.

Reduction Ideas: clause with more than 3 literals

1. Case clause with $k > 3$ literals: Let $c = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_k$. Let u be a new variable. Consider

$$c' = \Big(\ell_1 \vee \ell_2 \ldots \ell_{k-2} \vee u\Big) \wedge \Big(\ell_{k-1} \vee \ell_k \vee \neg u\Big).$$

2. Suppose $\varphi = \psi \wedge c$. Then $\varphi' = \psi \wedge c'$ is satisfiable iff φ is satisfiable.

Breaking a clause

Lemma

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

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

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

$$\left(X \vee z\right) \wedge \left(Y \vee \neg z\right) \text{ is satisfiable}$$

(with the same assignment to the variables appearing in X *and* Y*).*

SAT $\leq_P$ 3SAT (contd)

Clauses with more than 3 literals

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

$$
\begin{aligned}
c' = & \left(\ell_1 \vee \ell_2 \vee u_1\right) \wedge \left(\ell_3 \vee \neg u_1 \vee u_2\right) \\
& \wedge \left(\ell_4 \vee \neg u_2 \vee u_3\right) \wedge \\
& \cdots \wedge \left(\ell_{k-2} \vee \neg u_{k-4} \vee u_{k-3}\right) \wedge \left(\ell_{k-1} \vee \ell_k \vee \neg u_{k-3}\right).
\end{aligned}
$$

Claim

$\varphi = \psi \wedge c$ is satisfiable iff $\varphi' = \psi \wedge c'$ is satisfiable.

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

$$c' = \left(\ell_1 \vee \ell_2 \ldots \vee \ell_{k-2} \vee u_{k-3}\right) \wedge \left(\ell_{k-1} \vee \ell_k \vee \neg u_{k-3}\right).$$

An Example

Example

$$\varphi = \Big(\neg x_1 \lor \neg x_4\Big) \land \Big(x_1 \lor \neg x_2 \lor \neg x_3\Big)$$
$$\land \Big(\neg x_2 \lor \neg x_3 \lor x_4 \lor x_1\Big) \land \Big(x_1\Big).$$

Equivalent form:

$$\psi = (\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z)$$

An Example

Example

$$\varphi = \left(\neg x_1 \vee \neg x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \\ \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\right) \wedge \left(x_1\right).$$

Equivalent form:

$$\psi = (\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z) \\ \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$$

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:

$$\begin{aligned}\psi = &(\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z) \\ &\land (x_1 \lor \neg x_2 \lor \neg x_3) \\ &\land (\neg x_2 \lor \neg x_3 \lor y_1) \land (x_4 \lor x_1 \lor \neg y_1)\end{aligned}$$

An Example

Example

$$\varphi = \Big(\neg x_1 \vee \neg x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \\ \wedge \Big(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\Big) \wedge \Big(x_1\Big).$$

Equivalent form:

$$\begin{aligned}\psi = &(\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z) \\ &\wedge (x_1 \vee \neg x_2 \vee \neg x_3) \\ &\wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1) \\ &\wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v) \\ &\wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v).\end{aligned}$$

Overall Reduction Algorithm

Reduction from SAT to 3SAT

```
ReduceSATTo3SAT(φ):
    // φ: CNF formula.
    for each clause c of φ do
        if c does not have exactly 3 literals then
            construct c' as before
        else
            c' = c
    ψ is conjunction of all c' constructed in loop
    return Solver3SAT(ψ)
```

Correctness (informal)

φ is satisfiable iff ψ is satisfiable because for each clause c, the new $3\mathrm{CNF}$ 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 essense 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 φ_G such that G has a Hamilton Path iff φ_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 φ_G such that G has a Hamilton Path iff φ_G is satisfiable

Alternative view: Program/express using constraints

- What are variables?
- What are the constraints?

Reduce Directed Hamilton Path to SAT

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

Given G obtain CNF formula φ_G such that G has a Hamilton Path iff φ_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?

Reduction continued

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

Reduction continued

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 $\neg x(u, i) \vee \neg x(u, j)$ 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 $E(G)$

Reduction continued

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 $\neg x(u, i) \vee \neg x(u, j)$ 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 $E(G)$ $(x(u, i) \wedge x(v, i + 1)) \Rightarrow z(u, v)$ 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 (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

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 $(x_u \vee x_v)$. 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 $(x_u \vee x_v)$. 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

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

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

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**

Mathematical Programming

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

$$
\begin{array}{lll}
\text{maximize/minimize} & \sum_{j=1}^{n} c_j x_j & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots p \\
 & \sum_{j=1}^{n} a_{ij} x_j = b_i & \text{for } i = p+1 \ldots q \\
 & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \text{for } i = q+1 \ldots m
\end{array}
$$

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

$$
\begin{array}{lll}
\text{maximize/minimize} & \sum_{j=1}^{n} c_j x_j & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for } i = 1 \ldots p \\
 & \sum_{j=1}^{n} a_{ij} x_j = b_i & \text{for } i = p+1 \ldots q \\
 & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \text{for } i = q+1 \ldots n \\
 & x_i \in \mathbb{Z} & \text{for } i = 1 \text{ to } d
\end{array}
$$

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

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \le b_i \quad \text{for } i = 1 \ldots m \end{array}$$

$f, g_1, g_2, \ldots, g_m$ are convex functions

Mathematical Programming

- LP is a specical 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 specical 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
- ...