CMSC 451: SAT, Coloring, Hamiltonian Cycle, TSP

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Sects. 8.2, 8.7, 8.5 of Algorithm Design by Kleinberg & Tardos.
Boolean Formulas

**Variables:** $x_1, x_2, x_3$ (can be either true or false)

**Terms:** $t_1, t_2, \ldots, t_\ell$: $t_j$ is either $x_i$ or $\overline{x_i}$ (meaning either $x_i$ or not $x_i$).

**Clauses:** $t_1 \lor t_2 \lor \cdots \lor t_\ell$ ($\lor$ stands for “OR”)
A clause is true if any term in it is true.

**Example 1:** $(x_1 \lor \overline{x_2}), (\overline{x_1} \lor x_3), (x_2 \lor \overline{v_3})$

**Example 2:** $(x_1 \lor x_2 \lor \overline{x_3}), (\overline{x_2} \lor x_1)$
**Boolean Formulas**

**Def.** A truth assignment is a choice of **true** or **false** for each variable, ie, a function $\nu : X \rightarrow \{\text{true}, \text{false}\}$.

**Def.** A CNF formula is a conjunction of clauses:

$$C_1 \land C_2, \land \cdots \land C_k$$

**Example:** $(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_2 \lor \overline{v_3})$

**Def.** A truth assignment is a satisfying assignment for such a formula if it makes every clause **true**.
### Satisfiability (SAT)

Given a set of clauses $C_1, \ldots, C_k$ over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?

### Satisfiability (3-SAT)

Given a set of clauses $C_1, \ldots, C_k$, each of length 3, over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?
Cook-Levin Theorem

Theorem (Cook-Levin)

3-SAT is NP-complete.

Proven in early 1970s by Cook. Slightly different proof by Levin independently.

Idea of the proof: encode the workings of a Nondeterministic Turing machine for an instance $I$ of problem $X \in \text{NP}$ as a SAT formula so that the formula is satisfiable if and only if the nondeterministic Turing machine would accept instance $I$.

We won’t have time to prove this, but it gives us our first hard problem.
Reducing 3-SAT to Independent Set

**Thm.** 3-SAT $\leq_P$ Independent Set

*Proof.* Suppose we have an algorithm to solve Independent Set, how can we use it to solve 3-SAT?

To solve 3-SAT:

- you have to choose a term from each clause to set to **true**,
- but you can’t set both $x_i$ and $\bar{x}_i$ to **true**.

How do we do the reduction?
3-SAT \leq_P \text{ Independent Set}

\begin{align*}
(x_1 \lor x_2 \lor \overline{x_3}) & \land (x_2 \lor x_3 \lor \overline{x_4}) & \land (x_1 \lor \overline{x_2} \lor x_4)
\end{align*}

\begin{tikzpicture}
  \node[vertex] at (0,0) (x1) {$x_1$};
  \node[vertex] at (-2,-2) (x2) {$x_2$};
  \node[vertex] at (2,-2) (x3) {$x_3$};
  \node[vertex] at (-4,-4) (x4) {$x_4$};

  \node[vertex] at (-2,-4) (x5) {$\overline{x_2}$};
  \node[vertex] at (2,-4) (x6) {$\overline{x_3}$};
  \node[vertex] at (-6,-6) (x7) {$\overline{x_4}$};

  \node[vertex] at (-6,-8) (x8) {$\overline{x_2}$};
  \node[vertex] at (6,-8) (x9) {$\overline{x_2}$};

  \draw (x1) -- (x2);
  \draw (x1) -- (x3);
  \draw (x1) -- (x4);
  \draw (x2) -- (x3);
  \draw (x2) -- (x4);
  \draw (x3) -- (x4);
  \draw (x2) -- (x5);
  \draw (x3) -- (x6);
  \draw (x4) -- (x7);
  \draw (x5) -- (x6);
  \draw (x6) -- (x7);
  \draw (x7) -- (x8);
  \draw (x8) -- (x9);
  \draw (x9) -- (x7);
\end{tikzpicture}
Proof

Theorem

This graph has an independent set of size $k$ iff the formula is satisfiable.

Proof. $\implies$ If the formula is satisfiable, there is at least one true literal in each clause. Let $S$ be a set of one such true literal from each clause. $|S| = k$ and no two nodes in $S$ are connected by an edge.

$\implies$ If the graph has an independent set $S$ of size $k$, we know that it has one node from each “clause triangle.” Set those terms to true. This is possible because no 2 are negations of each other. $\blacksquare$
Graph Coloring
Graph Coloring Problem

Given a graph $G$, can you color the nodes with $\leq k$ colors such that the endpoints of every edge are colored differently?

**Notation:** A $k$-coloring is a function $f : V \rightarrow \{1, \ldots, k\}$ such that for every edge $\{u, v\}$ we have $f(u) \neq f(v)$.

If such a function exists for a given graph $G$, then $G$ is $k$-colorable.
How can we test if a graph has a 2-coloring?
How can we test if a graph has a 2-coloring?

Check if the graph is bipartite.

Unfortunately, for $k \geq 3$, the problem is NP-complete.

**Theorem**

3-Coloring is NP-complete.
3-Coloring $\in \textbf{NP}$: A valid coloring gives a certificate.

We will show that:

$$3\text{-SAT} \leq_P 3\text{-Coloring}$$

Let $x_1, \ldots, x_n, C_1, \ldots, C_k$ be an instance of 3-SAT.

We show how to use 3-Coloring to solve it.
Reduction from 3-SAT

We construct a graph $G$ that will be 3-colorable iff the 3-SAT instance is satisfiable.

For every variable $x_i$, create 2 nodes in $G$, one for $x_i$ and one for $\overline{x}_i$. Connect these nodes by an edge:

![Graph with nodes and edges]

Create 3 special nodes $T$, $F$, and $B$, joined in a triangle:
Connecting them up

Connect every variable node to B:
Properties:

• Each of $x_i$ and $\bar{x}_i$ must get different colors
• Each must be different than the color of B.
• B, T, and F must get different colors.

Hence, any 3-coloring of this graph defines a valid truth assignment!

Still have to constrain the truth assignments to satisfy the given clauses, however.
Connect Clause \((t_1, t_2, t_3)\) up like this:
Suppose Every Term Was False

What if every term in the clause was assigned the false color?
Connect Clause \((t_1, t_2, t_3)\) up like this:
Connect Clause \((t_1, t_2, t_3)\) up like this:
Connect Clause \((t_1, t_2, t_3)\) up like this:
Connect Clause \((t_1, t_2, t_3)\) up like this:
Suppose there is a 3-coloring.

Top node is colorable iff one of its terms gets the \textbf{true} color.

Suppose there is a 3-coloring.

We get a satisfying assignment by:

- Setting $x_i = \textbf{true}$ iff $v_i$ is colored the same as $T$

Let $C$ be any clause in the formula. At least 1 of its terms must be true, because if they were all false, we couldn’t complete the coloring (as shown above).
Suppose there is a satisfying assignment.

We get a 3-coloring of $G$ by:

- Coloring $T$, $F$, $B$ arbitrarily with 3 different colors
- If $x_i = \text{true}$, color $v_i$ with the same color as $T$ and $\bar{v}_i$ with the color of $F$.
- If $x_i = \text{false}$, do the opposite.
- Extend this coloring into the clause gadgets.

Hence: the graph is 3-colorable iff the formula it is derived from is satisfiable.
General Proof Strategy

General Strategy for Proving Something is NP-complete:

1. **Must show that** $X \in \text{NP}$. Do this by showing there is an certificate that can be efficiently checked.

2. **Look at some problems that are known to be NP-complete** (there are thousands), and choose one $Y$ that seems “similar” to your problem in some way.

3. **Show that** $Y \leq_P X$. 
One strategy for showing that $Y \leq_P X$ often works:

1. Let $I_Y$ be any instance of problem $Y$.

2. Show how to construct an instance $I_X$ of problem $X$ in polynomial time such that:
   
   - If $I_Y \in Y$, then $I_X \in X$
   - If $I_X \in X$, then $I_Y \in Y$
Hamiltonian Cycle Problem

**Hamiltonian Cycle**

Given a directed graph $G$, is there a cycle that visits every vertex exactly once?

Such a cycle is called a **Hamiltonian cycle**.
Theorem

Hamiltonian Cycle is NP-complete.

**Proof.** First, HamCycle ∈ NP. Why?

Second, we show 3-SAT ≤_P Hamiltonian Cycle.

Suppose we have a black box to solve Hamiltonian Cycle, how do we solve 3-SAT?

In other words: how do we encode an instance I of 3-SAT as a graph G such that I is satisfiable exactly when G has a Hamiltonian cycle.

Consider an instance I of 3-SAT, with variables x₁, . . . , xₙ and clauses C₁, . . . , Cₖ.
Reduction Idea (very high level):

• Create some graph structure (a “gadget”) that represents the variables

• And some graph structure that represents the clauses

• Hook them up in some way that encodes the formula

• Show that this graph has a Ham. cycle iff the formula is satisfiable.
Direction we travel along this chain represents whether to set the variable to **true** or **false**.

\[ x_i \]
Hooking in the Clauses

Add a new node for each clause:

\[ C_k \]

Connect it this way if \( x_i \) in \( C_k \)

\[ C_j \]

Connect it this way if \( x_i \) in \( C_k \)

Direction we travel along this chain represents whether to set the variable to true or false.
Connecting up the paths
Connecting up the paths
A Hamiltonian path encodes a truth assignment for the variables (depending on which direction each chain is traversed)

For there to be a Hamiltonian cycle, we have to visit every clause node

We can only visit a clause if we satisfy it (by setting one of its terms to true)

Hence, if there is a Hamiltonian cycle, there is a satisfying assignment
Hamiltonian Path

**Hamiltonian Path:** Does $G$ contain a path that visits every node exactly once?

How could you prove this problem is NP-complete?
Hamiltonian Path: Does $G$ contain a path that visits every node exactly once?

How could you prove this problem is NP-complete?

Reduce Hamiltonian Cycle to Hamiltonian Path.

Given instance of Hamiltonian Cycle $G$, choose an arbitrary node $v$ and split it into two nodes to get graph $G'$:

$\begin{array}{c}
\text{v} \\
\text{v''} \\
\text{v'}
\end{array}$

Now any Hamiltonian Path must start at $v'$ and end at $v''$. 
Hamiltonian Path

$G''$ has a Hamiltonian Path $\iff G$ has a Hamiltonian Cycle.

$\implies$ If $G''$ has a Hamiltonian Path, then the same ordering of nodes (after we glue $v'$ and $v''$ back together) is a Hamiltonian cycle in $G$.

$\impliedby$ If $G$ has a Hamiltonian Cycle, then the same ordering of nodes is a Hamiltonian path of $G'$ if we split up $v$ into $v'$ and $v''$. □

Hence, Hamiltonian Path is NP-complete.
Traveling Salesman Problem

Given $n$ cities, and distances $d(i, j)$ between each pair of cities, does there exist a path of length $\leq k$ that visits each city?

Notes:

- We have a distance between every pair of cities.
- In this version, $d(i, j)$ doesn’t have to equal $d(j, i)$.
- And the distances don’t have to obey the triangle inequality $(d(i, j) \leq d(i, k) + d(k, j)$ for all $i, j, k$).
TSP large instance

- TSP visiting 24,978 (all) cities in Sweden.
- Solved by David Applegate, Robert Bixby, Vašek Chvátal, William Cook, and Keld Helsgaun
- [http://www.tsp.gatech.edu/sweden/index.html](http://www.tsp.gatech.edu/sweden/index.html)
- Lots more cool TSP at [http://www.tsp.gatech.edu/](http://www.tsp.gatech.edu/)
Thm. Traveling Salesman is NP-complete.

TSP seems a lot like Hamiltonian Cycle. We will show that

Hamiltonian Cycle $\leq_P TSP$

To do that:

Given: a graph $G = (V, E)$ that we want to test for a Hamiltonian cycle,

Create: an instance of TSP.
A TSP instance $D$ consists of $n$ cities, and $n(n - 1)$ distances.

**Cities** We have a city $c_i$ for every node $v_i$.

**Distances** Let $d(c_i, c_j) = \begin{cases} 1 & \text{if edge } (v_i, v_j) \in E \\ 2 & \text{otherwise} \end{cases}$
Proof. If $G$ has a Hamiltonian cycle, then this ordering of cities gives a tour of length $\leq n$ in $D$ (only distances of length 1 are used).

Suppose $D$ has a tour of length $\leq n$. The tour length is the sum of $n$ terms, meaning each term must equal 1, and hence cities that are visited consecutively must be connected by an edge in $G$. □

Also, TSP $\in$ NP: a certificate is simply an ordering of the $n$ cities.
Hence, TSP is NP-complete.

Even TSP restricted to the case when the $d(i,j)$ values come from actual distances on a map is NP-complete.