Chapter 23

More NP-Complete Problems

CS 473: Fundamental Algorithms, Spring 2013
April 19, 2013

23.0.0.1 Recap

**NP**: languages that have polynomial time certifiers/verifiers
A language $L$ is NP-Complete iff

- $L$ is in NP
- for every $L'$ in NP, $L' \leq_P L$

$L$ is NP-Hard if for every $L'$ in NP, $L' \leq_P L$.

**Theorem 23.0.1 (Cook-Levin).** Circuit-SAT and SAT are NP-Complete.

23.0.0.2 Recap contd

**Theorem 23.0.2 (Cook-Levin).** Circuit-SAT and SAT are NP-Complete.

Establish NP-Completeness via reductions:

- SAT $\leq_P$ 3-SAT and hence 3-SAT is NP-complete
- 3-SAT $\leq_P$ Independent Set (which is in NP) and hence Independent Set is NP-Complete
- Vertex Cover is NP-Complete
- Clique is NP-Complete
- Set Cover is NP-Complete
23.0.0.3 Today

Prove

- Hamiltonian Cycle Problem is NP-Complete
- 3-Coloring is NP-Complete

### 23.1 NP-Completeness of Hamiltonian Cycle

#### 23.1.1 Reduction from 3SAT to Hamiltonian Cycle

**23.1.1.1 Directed Hamiltonian Cycle**

**Input** Given a directed graph $G = (V, E)$ with $n$ vertices

**Goal** Does $G$ have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in $G$ exactly once

**23.1.1.2 Directed Hamiltonian Cycle is NP-Complete**

- Directed Hamiltonian Cycle is in $NP$
  - **Certificate:** Sequence of vertices
  - **Certifier:** Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge

- **Hardness:** We will show $3$-SAT $\leq_P$ Directed Hamiltonian Cycle

#### 23.1.1.3 Reduction

Given 3-SAT formula $\varphi$ create a graph $G_\varphi$ such that

- $G_\varphi$ has a Hamiltonian cycle if and only if $\varphi$ is satisfiable
- $G_\varphi$ should be constructible from $\varphi$ by a polynomial time algorithm $\mathcal{A}$

**Notation:** $\varphi$ has $n$ variables $x_1, x_2, \ldots, x_n$ and $m$ clauses $C_1, C_2, \ldots, C_m$. 

23.1.1.4 Reduction: First Ideas

- Viewing SAT: Assign values to \( n \) variables, and each clauses has 3 ways in which it can be satisfied.

- Construct graph with \( 2^n \) Hamiltonian cycles, where each cycle corresponds to some boolean assignment.

- Then add more graph structure to encode constraints on assignments imposed by the clauses.

23.1.1.5 The Reduction: Phase I

- Traverse path \( i \) from left to right iff \( x_i \) is set to true

- Each path has \( 3(m + 1) \) nodes where \( m \) is number of clauses in \( \varphi \); nodes numbered from left to right (1 to \( 3m + 3 \))

23.1.1.6 The Reduction: Phase II

- Add vertex \( c_j \) for clause \( C_j \). \( c_j \) has edge from vertex 3\( j \) and to vertex 3\( j + 1 \) on path \( i \) if \( x_i \) appears in clause \( C_j \), and has edge from vertex 3\( j + 1 \) and to vertex 3\( j \) if \( \neg x_i \) appears in \( C_j \).
23.1.1.7 Correctness Proof

Proposition 23.1.1. \( \varphi \) has a satisfying assignment iff \( G_\varphi \) has a Hamiltonian cycle.

Proof:

\( \Rightarrow \) Let \( a \) be the satisfying assignment for \( \varphi \). Define Hamiltonian cycle as follows

- If \( a(x_i) = 1 \) then traverse path \( i \) from left to right
- If \( a(x_i) = 0 \) then traverse path \( i \) from right to left
- For each clause, path of at least one variable is in the “right” direction to splice in the node corresponding to clause

\( \blacksquare \)

23.1.1.8 Hamiltonian Cycle \( \Rightarrow \) Satisfying assignment

Suppose \( \Pi \) is a Hamiltonian cycle in \( G_\varphi \)

- If \( \Pi \) enters \( c_j \) (vertex for clause \( C_j \)) from vertex \( 3j \) on path \( i \) then it must leave the clause vertex on edge to \( 3j + 1 \) on the same path \( i \)
  - If not, then only unvisited neighbor of \( 3j + 1 \) on path \( i \) is \( 3j + 2 \)
  - Thus, we don’t have two unvisited neighbors (one to enter from, and the other to leave) to have a Hamiltonian Cycle

- Similarly, if \( \Pi \) enters \( c_j \) from vertex \( 3j + 1 \) on path \( i \) then it must leave the clause vertex \( c_j \) on edge to \( 3j \) on path \( i \)
23.1.1.9 Example

23.1.1.10 Hamiltonian Cycle $\Rightarrow$ Satisfying assignment (contd)

- Thus, vertices visited immediately before and after $C_i$ are connected by an edge
- We can remove $c_j$ from cycle, and get Hamiltonian cycle in $G - c_j$
- Consider Hamiltonian cycle in $G - \{c_1, \ldots c_m\}$. It traverses each path in only one direction, which determines the truth assignment

23.1.2 Hamiltonian cycle in undirected graph

23.1.2.1 Hamiltonian Cycle

Problem 23.1.2. Input Given undirected graph $G = (V, E)$

Goal Does $G$ have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?

23.1.2.2 NP-Completeness

Theorem 23.1.3. Hamiltonian cycle problem for undirected graphs is NP-Complete.

Proof:

- The problem is in NP; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem
23.1.2.3 Reduction Sketch

**Goal:** Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}, v$, and $v_{out}$
- A directed edge $(a, b)$ is replaced by edge $(a_{out}, b_{in})$

![Diagram](image)

23.1.2.4 Reduction: Wrapup

- The reduction is polynomial time (exercise)
- The reduction is correct (exercise)

23.2 NP-Completeness of Graph Coloring

23.2.0.5 Graph Coloring

**Problem:** Graph Coloring

**Instance:** $G = (V, E)$: Undirected graph, integer $k$.

**Question:** Can the vertices of the graph be colored using $k$ colors so that vertices connected by an edge do not get the same color?

23.2.0.6 Graph 3-Coloring

**Problem:** 3 Coloring

**Instance:** $G = (V, E)$: Undirected graph.

**Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?
23.2.0.7 Graph Coloring

Observation: If $G$ is colored with $k$ colors then each color class (nodes of same color) form an independent set in $G$. Thus, $G$ can be partitioned into $k$ independent sets iff $G$ is $k$-colorable.

Graph 2-Coloring can be decided in polynomial time.

$G$ is 2-colorable iff $G$ is bipartite! There is a linear time algorithm to check if $G$ is bipartite using BFS (we saw this earlier).

23.2.1 Problems related to graph coloring

23.2.1.1 Graph Coloring and Register Allocation

Register Allocation Assign variables to (at most) $k$ registers such that variables needed at the same time are not assigned to the same register. Interference Graph Vertices are variables, and there is an edge between two vertices, if the two variables are “live” at the same time. Observations

- [Chaitin] Register allocation problem is equivalent to coloring the interference graph with $k$ colors
- Moreover, $3$-COLOR $\leq_p k$-Register Allocation, for any $k \geq 3$

23.2.1.2 Class Room Scheduling

Given $n$ classes and their meeting times, are $k$ rooms sufficient?

Reduce to Graph $k$-Coloring problem

Create graph $G$
- a node $v_i$ for each class $i$
- an edge between $v_i$ and $v_j$ if classes $i$ and $j$ conflict

Exercise: $G$ is $k$-colorable iff $k$ rooms are sufficient

23.2.1.3 Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint bands of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

Problem: given $k$ bands and some region with $n$ towers, is there a way to assign the bands to avoid interference?

Can reduce to $k$-coloring by creating interference/conflict graph on towers.
23.2.2 Showing hardness of \textbf{3 COLORING}

23.2.2.1 3-Coloring is NP-Complete

- \textbf{3-Coloring} is in \textbf{NP}.
  
  \begin{itemize}
    \item \textbf{Certificate}: for each node a color from \{1, 2, 3\}.
    \item \textbf{Certifier}: Check if for each edge \((u, v)\), the color of \(u\) is different from that of \(v\).
  \end{itemize}

- \textbf{Hardness}: We will show \(3\)-\text{SAT} \(\leq_P 3\)-\text{Coloring}.

23.2.2.2 Reduction Idea

Start with \textbf{3SAT} formula (i.e., 3CNF formula) \(\varphi\) with \(n\) variables \(x_1, \ldots, x_n\) and \(m\) clauses \(C_1, \ldots, C_m\). Create graph \(G_{\varphi}\) such that \(G_{\varphi}\) is 3-colorable iff \(\varphi\) is satisfiable.

- \textbf{i+}: need to establish truth assignment for \(x_1, \ldots, x_n\) via colors for some nodes in \(G_{\varphi}\).

- \textbf{i+}: create triangle with node True, False, Base

- \textbf{i+}: for each variable \(x_i\) two nodes \(v_i\) and \(\overline{v_i}\) connected in a triangle with common Base

- \textbf{i+}: If graph is 3-colored, either \(v_i\) or \(\overline{v_i}\) gets the same color as True. Interpret this as a truth assignment to \(v_i\)

- \textbf{i+}: Need to add constraints to ensure clauses are satisfied (next phase)

23.2.2.3 Figure
23.2.2.4 Clause Satisfiability Gadget

For each clause \( C_j = (a \lor b \lor c) \), create a small gadget graph

- gadget graph connects to nodes corresponding to \( a, b, c \)
- needs to implement OR

OR-gadget-graph:

![OR-gadget-graph](image)

23.2.2.5 OR-Gadget Graph

**Property:** if \( a, b, c \) are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

**Property:** if one of \( a, b, c \) is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

23.2.2.6 Reduction

- create triangle with nodes True, False, Base
- for each variable \( x_i \) two nodes \( v_i \) and \( \bar{v}_i \) connected in a triangle with common Base
- for each clause \( C_j = (a \lor b \lor c) \), add OR-gadget graph with input nodes \( a, b, c \) and connect output node of gadget to both False and Base

![Reduction](image)

23.2.2.7 Reduction

![Reduction](image)
Claim 23.2.1. No legal 3-coloring of above graph (with coloring of nodes T, F, B fixed) in which a, b, c are colored False. If any of a, b, c are colored True then there is a legal 3-coloring of above graph.

23.2.2.8 3 coloring of the clause gadget

23.2.2.9 Reduction Outline

Example 23.2.2. $\varphi = (u \lor \neg v \lor w) \land (v \lor x \lor \neg y)$

23.2.2.10 Correctness of Reduction

$\varphi$ is satisfiable implies $G_\varphi$ is 3-colorable

$\vdash +$ if $x_i$ is assigned True, color $v_i$ True and $\bar{v}_i$ False
\( i+\phi \) for each clause \( C_j = (a \lor b \lor c) \) at least one of \( a, b, c \) is colored True. OR-gadget for \( C_j \) can be 3-colored such that output is True.

\( G_\varphi \) is 3-colorable implies \( \varphi \) is satisfiable

\( i+\phi \) if \( v_i \) is colored True then set \( x_i \) to be True, this is a legal truth assignment

\( i+\phi \) consider any clause \( C_j = (a \lor b \lor c) \). it cannot be that all \( a, b, c \) are False. If so, output of OR-gadget for \( C_j \) has to be colored False but output is connected to Base and False!

23.2.3 Graph generated in reduction...

23.2.3.1 ... from 3SAT to 3COLOR

23.3 Hardness of **Subset Sum**

23.3.0.2 Subset Sum

Problem: **Subset Sum**

| Instance: \( S \) - set of positive integers, \( t \): an integer number (Target) |
| Question: Is there a subset \( X \subseteq S \) such that \( \sum_{x \in X} x = t \)? |

Claim 23.3.1. **Subset Sum** is NP-Complete.
23.3.0.3 Vec Subset Sum

We will prove following problem is NP-Complete...

Problem: Vec Subset Sum

<table>
<thead>
<tr>
<th>Instance:</th>
<th>( S ) - set of ( n ) vectors of dimension ( k ), each vector has non-negative numbers for its coordinates, and a target vector ( \vec{t} ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question:</td>
<td>Is there a subset ( X \subseteq S ) such that ( \sum_{\vec{x} \in X} \vec{x} = \vec{t} )?</td>
</tr>
</tbody>
</table>

Reduction from 3SAT.

23.3.1 Vec Subset Sum

23.3.1.1 Handling a single clause

Think about vectors as being lines in a table.

First gadget

Selecting between two lines.

<table>
<thead>
<tr>
<th>Target</th>
<th>??</th>
<th>??</th>
<th>01</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>??</td>
<td>??</td>
<td>01</td>
<td>??</td>
</tr>
</tbody>
</table>

Two rows for every variable \( x \): selecting either \( x = 0 \) or \( x = 1 \).

23.3.1.2 Handling a clause...

We will have a column for every clause...

<table>
<thead>
<tr>
<th>numbers</th>
<th>??</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( \overline{a} )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( b )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( \overline{b} )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( c )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( \overline{c} )</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>( C ) fix-up 1</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>( C ) fix-up 2</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>( C ) fix-up 3</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>( \text{TARGET} )</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

...
### 23.3.1.3 3SAT to Vec Subset Sum

<table>
<thead>
<tr>
<th>numbers</th>
<th>$a \lor \bar{a}$</th>
<th>$b \lor \bar{b}$</th>
<th>$c \lor \bar{c}$</th>
<th>$d \lor \bar{d}$</th>
<th>$D \equiv \bar{a} \lor \bar{b} \lor \bar{c}$</th>
<th>$C \equiv a \lor b \lor c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>$\bar{6}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>$\bar{5}$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$\bar{4}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$C_{\text{fix-up 1}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$C_{\text{fix-up 2}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$D_{\text{fix-up 1}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$D_{\text{fix-up 2}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$D_{\text{fix-up 3}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>TARGET</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

### 23.3.1.4 Vec Subset Sum to Subset Sum

<table>
<thead>
<tr>
<th>numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000000001</td>
</tr>
<tr>
<td>010000000000</td>
</tr>
<tr>
<td>000100000001</td>
</tr>
<tr>
<td>000100000100</td>
</tr>
<tr>
<td>000001000100</td>
</tr>
<tr>
<td>000001000001</td>
</tr>
<tr>
<td>000000100000</td>
</tr>
<tr>
<td>000000010101</td>
</tr>
<tr>
<td>000000000007</td>
</tr>
<tr>
<td>000000000008</td>
</tr>
<tr>
<td>000000000009</td>
</tr>
<tr>
<td>000000000070</td>
</tr>
<tr>
<td>000000000080</td>
</tr>
<tr>
<td>000000000090</td>
</tr>
<tr>
<td>010101011010</td>
</tr>
</tbody>
</table>

### 23.3.1.5 Other NP-Complete Problems

- 3-Dimensional Matching
- Subset Sum

Read book.

### 23.3.1.6 Need to Know NP-Complete Problems

- 3-SAT
- Circuit-SAT
- Independent Set
• Vertex Cover
• Clique
• Set Cover
• Hamiltonian Cycle in Directed/Undirected Graphs
• 3-Coloring
• 3-D Matching
• Subset Sum

23.3.1.7 Subset Sum and Knapsack

Subset Sum Problem: Given \( n \) integers \( a_1, a_2, \ldots, a_n \) and a target \( B \), is there a subset of \( S \) of \( \{a_1, \ldots, a_n\} \) such that the numbers in \( S \) add up precisely to \( B \)?

Subset Sum is \textbf{NP-Complete}—see book.

Knapsack: Given \( n \) items with item \( i \) having size \( s_i \) and profit \( p_i \), a knapsack of capacity \( B \), and a target profit \( P \), is there a subset \( S \) of items that can be packed in the knapsack and the profit of \( S \) is at least \( P \)?

Show Knapsack problem is \textbf{NP-Complete} via reduction from Subset Sum (exercise).

23.3.1.8 Subset Sum and Knapsack

Subset Sum can be solved in \( O(nB) \) time using dynamic programming (exercise). Implies that problem is hard only when numbers \( a_1, a_2, \ldots, a_n \) are exponentially large compared to \( n \). That is, each \( a_i \) requires polynomial in \( n \) bits.

\textit{Number problems} of the above type are said to be \textit{weakly NPComplete}. 

16
Bibliography