CS21
Decidability and Tractability
Lecture 20
February 24, 2017
Outline

• NP complete problems
  – NP-complete problems: Hamilton path and cycle, Traveling Salesperson Problem
Hamilton Path

• Definition: given a directed graph $G = (V, E)$, a **Hamilton path** in $G$ is a directed path that touches every node exactly once.

• A language (decision problem):
  $\text{HAMPATH} = \{ (G, s, t) : G \text{ has a Hamilton path from } s \text{ to } t \}$
HAMPATH is NP-complete

**Theorem**: the following language is NP-complete:

\[ \text{HAMPATH} = \{(G, s, t) : G \text{ has a Hamilton path from } s \text{ to } t\} \]

• Proof:
  – Part 1: HAMPATH ∈ NP. Proof?
  – Part 2: HAMPATH is NP-hard.
    • reduce from?
HAMPATH is NP-complete

- We are reducing from the language:

\[ 3\text{SAT} = \{ \phi : \phi \text{ is a 3-CNФ formula that has a satisfying assignment } \} \]

to the language:

\[ \text{HAMPATH} = \{ (G, s, t) : G \text{ has a Hamilton path from } s \text{ to } t \} \]
HAMPATH is NP-complete

• We want to construct a graph from $\varphi$ with the following properties:
  – a satisfying assignment to $\varphi$ translates into a Hamilton Path from $s$ to $t$
  – a Hamilton Path from $s$ to $t$ can be translated into a satisfying assignment for $\varphi$

• We will build the graph up from pieces called gadgets that “simulate” the clauses and variables of $\varphi$. 
HAMPATH is NP-complete

• The variable gadget (one for each $x_i$):
HAMPATH is NP-complete

- path from $s$ to $t$ translates into a truth assignment to $x_1...x_m$

- why must the path be of this form?
HAMPATH is NP-complete

\[ \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \land (\ldots) \]

• How to ensure that all \( k \) clauses are satisfied?
• need to add nodes
  – can be visited in path if the clause is satisfied
  – if visited in path, implies clause is satisfied by the assignment given by path through variable gadgets
HAMPATH is NP-complete

- \( \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \land (\ldots) \)
- Clause gadget allows “detour” from “assignment path” for each true literal in clause
HAMPATH is NP-complete

• One clause gadget for each of k clauses:

```
for clause 1
```

```
for clause 2
```

"x_1" [diagram]

"x_2" [diagram]

"x_m" [diagram]

"C_1" [diagram]

"C_2" [diagram]

⋮ [diagram]

⋮ [diagram]

"C_k" [diagram]
HAMPATH is NP-complete

\[ \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \]

- \( f(\varphi) \) is this graph (edges to/from clause nodes not pictured)
- \( f \) poly-time computable?
- # nodes = \( O(\text{km}) \)

\( x_1 \)  \( x_2 \)  \( x_m \)

\( C_1 \)  \( C_2 \)  \( \cdots \)  \( C_k \)
HAMPATH is NP-complete

\[ \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_5) \land \ldots \]

\( \bullet \) YES maps to \( \bullet \) YES?

\( \text{\textbullet{C}}_1 \) • first form path from satisfying assign.

\( \text{\textbullet{C}}_2 \) • pick true literal in each clause and add detour

\( \vdots \)

\( \text{\textbullet{C}}_k \)
HAMPATH is NP-complete

\[ \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \]

- NO maps to NO?
- try to translate path into satisfying assignment
- if path has “intended” form, OK.
HAMPATH is NP-complete

• What can go wrong?
  – path has “intended form” unless return from clause gadget to different variable gadget

we will argue that this cannot happen:
HAMPATH is NP-complete

Case 1 (positive occurrence of v in clause):

- path must visit y
- must enter from x, z, or c
- must exit to z (x is taken)
- x, c are taken. can’t happen
HAMPATH is NP-complete

Case 2 (negative occurrence of v in clause):

- path must visit y
- must enter from x or z
- must exit to z
- x is taken
- can’t happen

“V”
Undirected Hamilton Path

• HAMPATH refers to a directed graph.
• Is it easier on an undirected graph?

• A language (decision problem):
  \[ UHAMPATH = \{ (G, s, t) : \text{undirected } G \text{ has a Hamilton path from } s \text{ to } t \} \]
UHAMPATH is NP-complete

**Theorem:** the following language is NP-complete:

\[ \text{UHAMPATH} = \{(G, s, t) : \text{undirected graph } G \text{ has a Hamilton path from } s \text{ to } t\} \]

- **Proof:**
  - Part 1: UHAMPATH $\in$ NP. Proof?
  - Part 2: UHAMPATH is NP-hard.
    - reduce from?
UHAMPATH is NP-complete

• We are reducing from the language:

\[ \text{HAMPATH} = \{(G, s, t) : \text{directed graph } G \text{ has a Hamilton path from } s \text{ to } t\} \]

to the language:

\[ \text{UHAMPATH} = \{(G, s, t) : \text{undirected graph } G \text{ has a Hamilton path from } s \text{ to } t\} \]
UHAMPATH is NP-complete

- The reduction:

  - replace each node with three (except s, t)
  - \((u_{in}, u_{mid})\)
  - \((u_{mid}, u_{out})\)
  - \((u_{out}, v_{in})\) iff G has (u, v)

\[\begin{align*}
\text{G} & \quad \rightarrow \quad \text{G'} \\
\text{s} & \quad \text{u} \quad \text{v} \quad \text{t} \\
\text{u} & \quad \text{v} \\
\end{align*}\]
UHAMPATH is NP-complete

- Does the reduction run in poly-time?

- YES maps to YES?
  - Hamilton path in G: $s, u_1, u_2, u_3, \ldots, u_k, t$
  - Hamilton path in G’: 
    - $s_{\text{out}}, (u_1)_{\text{in}}, (u_1)_{\text{mid}}, (u_1)_{\text{out}}, (u_2)_{\text{in}}, (u_2)_{\text{mid}}, (u_2)_{\text{out}}, \ldots$
    - $(u_k)_{\text{in}}, (u_k)_{\text{mid}}, (u_k)_{\text{out}}, t_{\text{in}}$
UHAMPATH is NP-complete

• NO maps to NO?
  – Hamilton path in $G'$:
    
    \[ s_{\text{out}}, v_1, v_2, v_3, v_4, v_5, v_6, \ldots, v_{k-2}, v_{k-1}, v_k, t_{\text{in}} \]
    
    – $v_1 = (u_{i_1})_{\text{in}}$ for some $i_1$ (only edges to ins)
    – $v_2 = (u_{i_1})_{\text{mid}}$ for some $i_1$ (only way to enter mid)
    – $v_3 = (u_{i_1})_{\text{out}}$ for some $i_1$ (only way to exit mid)
    – $v_4 = (u_{i_2})_{\text{in}}$ for some $i_2$ (only edges to ins)
    – ...
    
    – Hamilton path in $G$: $s, u_{i_1}, u_{i_2}, u_{i_3}, \ldots, u_{i_k}, t$
Undirected Hamilton Cycle

• Definition: given a undirected graph $G = (V, E)$, a Hamilton cycle in $G$ is a cycle in $G$ that touches every node exactly once.

• Is finding one easier than finding a Hamilton path?

• A language (decision problem):
  $UHAMCYCLE = \{G : G \text{ has a Hamilton cycle}\}$
UHAMCYCLE is NP-complete

**Theorem**: the following language is NP-complete:

\[ \text{UHAMCYCLE} = \{G: G \text{ has a Hamilton cycle} \} \]

• Proof:
  – Part 1: UHAMCYCLE \( \in \) NP. Proof?
  – Part 2: UHAMCYCLE is NP-hard.
    • reduce from?
UHAMCYCLE is NP-complete

• The reduction (from UHAMPATH):

- H. path from s to t implies H. cycle in G'
- H. cycle in G’ must visit u via red edges
- removing red edges gives H. path from s to t in G
Traveling Salesperson Problem

• Definition: given n cities $v_1, v_2, \ldots, v_n$ and inter-city distances $d_{i,j}$ a TSP tour in G is a permutation $\pi$ of $\{1\ldots n\}$. The tour’s length is $\Sigma_{i=1\ldots n} d_{\pi(i), \pi(i+1)}$ (where n+1 means 1).

• A search problem:
  given the $\{d_{i,j}\}$, find the shortest TSP tour

• corresponding language (decision problem):
  $\text{TSP} = \{(\{d_{i,j} : 1 \leq i < j \leq n\}, k) : \text{these cities have a TSP tour of length} \leq k\}$
TSP is NP-complete

**Theorem**: the following language is NP-complete:

\[
TSP = \{(\{d_{i,j} : 1 \leq i < j \leq n\}, k) : \text{these cities have a TSP tour of length } \leq k\}
\]

• Proof:
  – Part 1: TSP ∈ NP. Proof?
  – Part 2: TSP is NP-hard.
    • reduce from?
TSP is NP-complete

• We are reducing from the language:

\[ \text{UHAMCYCLE} = \{G : G \text{ has a Hamilton cycle}\} \]

to the language:

\[ \text{TSP} = \{(\{d_{i,j} : 1 \leq i<j \leq n\}, k) : \text{these cities have a TSP tour of length } \leq k\} \]
TSP is NP-complete

• The reduction:
  – given $G = (V, E)$ with $n$ nodes
  produce:
  – $n$ cities corresponding to the $n$ nodes
  – $d_{u,v} = 1$ if $(u, v) \in E$
  – $d_{u,v} = 2$ if $(u, v) \notin E$
  – set $k = n$
TSP is NP-complete

- YES maps to YES?
  - if $G$ has a Hamilton cycle, then visiting cities in that order gives TSP tour of length $n$

- NO maps to NO?
  - if TSP tour of length $\leq n$, it must have length exactly $n$.
  - all distances in tour are 1. Must be edges between every successive pair of cities in tour.
Subset Sum

• A language (decision problem):
  \[\text{SUBSET-SUM} = \{(S = \{a_1, a_2, a_3, \ldots, a_k\}, B) : \text{there is a subset of } S \text{ that sums to } B\}\]

• example:
  – \(S = \{1, 7, 28, 3, 2, 5, 9, 32, 41, 11, 8\}\)
  – \(B = 30\)
  – \(30 = 7 + 3 + 9 + 11. \text{yes.}\)
Subset Sum

$\text{SUBSET-SUM} = \{(S = \{a_1, a_2, a_3, \ldots, a_k\}, B) : \text{there is a subset of } S \text{ that sums to } B\}$

• Is this problem NP-complete? in P?

• Problem set: in $\text{TIME}(B \notin \text{poly}(k))$
SUBSET-SUM is NP-complete

**Theorem**: the following language is NP-complete:

$\text{SUBSET-SUM} = \{(S, B) : \text{there is a subset of } S \text{ that sums to } B\}$

• Proof:
  – Part 1: SUBSET-SUM $\in$ NP. Proof?
  – Part 2: SUBSET-SUM is NP-hard.
    • reduce from?

our reduction had better produce super-polynomially large $B$ (unless we want to prove $P=NP$)
SUBSET-SUM is NP-complete

• We are reducing from the language:

\[ 3 \text{SAT} = \{ \phi : \phi \text{ is a 3-CNF formula that has a satisfying assignment} \} \]

to the language:

\[ \text{SUBSET-SUM} = \{ (S = \{a_1, a_2, a_3, \ldots, a_k\}, B) : \text{there is a subset of } S \text{ that sums to } B \} \]
SUBSET-SUM is NP-complete

• \( \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \land (\ldots) \)

• Need integers to play the role of truth assignments

• For each variable \( x_i \) include two integers in our set \( S \):
  – \( x_i^{\text{TRUE}} \) and \( x_i^{\text{FALSE}} \)

• Set \( B \) so that exactly one must be in sum
SUBSET-SUM is NP-complete

\[
\begin{align*}
  x_1^{\text{TRUE}} &= 1 \ 0 \ 0 \ 0 \ \ldots \ 0 \\
  x_1^{\text{FALSE}} &= 1 \ 0 \ 0 \ 0 \ \ldots \ 0 \\
  x_2^{\text{TRUE}} &= 0 \ 1 \ 0 \ 0 \ \ldots \ 0 \\
  x_2^{\text{FALSE}} &= 0 \ 1 \ 0 \ 0 \ \ldots \ 0 \\
  \ldots \\
  x_m^{\text{TRUE}} &= 0 \ 0 \ 0 \ 0 \ \ldots \ 1 \\
  x_m^{\text{FALSE}} &= 0 \ 0 \ 0 \ 0 \ \ldots \ 1 \\
  B &= 1 \ 1 \ 1 \ 1 \ \ldots \ 1
\end{align*}
\]

- every choice of one from each \((x_i^{\text{TRUE}}, x_i^{\text{FALSE}})\) pair sums to \(B\)
- every subset that sums to \(B\) must choose one from each \((x_i^{\text{TRUE}}, x_i^{\text{FALSE}})\) pair
SUBSET-SUM is NP-complete

• \( \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \land (\ldots) \)

• Need to force subset to “choose” at least one true literal from each clause

• Idea:
  – add more digits
  – one digit for each clause
  – set B to force each clause to be satisfied.
SUBSET-SUM is NP-complete

\[ \neg \varphi = (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_4 \lor x_3) \land \ldots \land (\ldots) \]

<table>
<thead>
<tr>
<th>Variable</th>
<th>True</th>
<th>False</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(1000\ldots01)</td>
<td>(1000\ldots00)</td>
<td>(0100\ldots01)</td>
<td>(0100\ldots00)</td>
</tr>
<tr>
<td>(x_2)</td>
<td>(0100\ldots01)</td>
<td>(0100\ldots00)</td>
<td>(0010\ldots00)</td>
<td>(0010\ldots01)</td>
</tr>
<tr>
<td>(x_3)</td>
<td>(0010\ldots00)</td>
<td>(0010\ldots01)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
</tbody>
</table>

\(B = 1111\ldots1???)

clause 1

clause 2

clause 3

\(\vdots\)

clause k
SUBSET-SUM is NP-complete

– B = 1 1 1 1 ... 1 ? ? ? ?
– if clause i is satisfied sum might be 1, 2, or 3 in corresponding column.
– want ? to “mean” ≥ 1
– solution: set ? = 3
– add two “filler” elements for each clause i:
  – FILL1_i = 0 0 0 0 ... 0 0 ... 0 1 0 ... 0
  – FILL2_i = 0 0 0 0 ... 0 0 ... 0 1 0 ... 0

`column for clause i`
SUBSET-SUM is NP-complete

• Reduction: m variables, k clauses
  – for each variable $x_i$:
    • $x_i^{\text{TRUE}}$ has ones in positions $k + i$ and $\{j : \text{clause } j \text{ includes literal } x_i\}$
    • $x_i^{\text{FALSE}}$ has ones in positions $k + i$ and $\{j : \text{clause } j \text{ includes literal } \neg x_i\}$
  – for each clause $i$:
    • $\text{FILL1}_i$ and $\text{FILL2}_i$ have one in position $i$
  – bound $B$ has 3 in positions 1…$k$ and 1 in positions $k+1$…$k+m$
SUBSET-SUM is NP-complete

- Reduction computable in poly-time?
- YES maps to YES?
  - choose one from each \((x_i^{\text{TRUE}}, x_i^{\text{FALSE}})\) pair corresponding to a satisfying assignment
  - choose 0, 1, or 2 of filler elements for each clause \(i\) depending on whether it has 3, 2, or 1 true literals
  - first \(m\) digits add to 1; last \(k\) digits add to 3
SUBSET-SUM is NP-complete

• NO maps to NO?
  – at most 5 ones in each column, so no carries to worry about
  – first m digits of B force subset to choose exactly one from each \((x_i^{\text{TRUE}}, x_i^{\text{FALSE}})\) pair
  – last k digits of B require at least one true literal per clause, since can only sum to 2 using filler elements
  – resulting assignment must satisfy \(\varphi\)