NP-Completeness

Sections 28.5, 28.6
NP-Completeness

A language $L$ might have these properties:

1. $L$ is in NP.
2. Every language in NP is deterministic, polynomial-time reducible to $L$.

- $L$ is **NP-hard** iff it possesses property 2.
  
  An NP-hard language is at least as hard as any other language in NP.

- $L$ is **NP-complete** iff it possesses *both* property 1 and property 2.
  
  All NP-complete languages can be viewed as being equivalently hard.
NP-Hard vs. NP-Complete

An example: puzzles vs. games (Appendix N).

To use this theory to analyze a game like chess, we must generalize it so that we can talk about solution time as a function of problem size:

CHESS = \{<b>\}: \ b \text{ is a configuration of an } n \times n \text{ chess board and there is a guaranteed win for the current player}\}.
Sudoku

- **SUDOKU** = \{<b>: b is a configuration of an $n \times n$ grid and b has a solution under the rules of Sudoku\}.
Sudoku

- **SUDOKU** = \{<b>: b is a configuration of an \( n \times n \) grid and b has a solution under the rules of Sudoku}\}.

A deterministic, polynomial time verifier for SUDOKU, on input:

\(<b, (1,1,1), (1,2,5), (1,3,4), \ldots>\)
Chess

A deterministic polynomial time verifier for CHESS?
NP-Hard vs. NP-Complete

SUDOKU = \{<b>: b is a configuration of an \( n \times n \) grid and b has a solution under the rules of Sudoku}\}.

\textit{NP-complete.}

CHESS = \{<b>: b is a configuration of an \( n \times n \) chess board and there is a guaranteed win for the current player}\}.

\textit{NP-hard, not thought to be in NP.}
\textit{If fixed number of pieces: PSPACE-complete.}
\textit{If varialbe number of pieces: EXPTIME-complete.}
Showing that $L$ is NP-Complete

How about: Take a list of known NP languages and crank out the reductions?

$NPL_1 \geq L$

$NPL_2 \geq L$

$NPL_3 \geq L$

…
Showing that \( L \) is NP-Complete

Suppose we had one NP-complete language \( L' \):

\[
\begin{align*}
NPL_1 & \quad NPL_2 \quad NPL_3 \quad NPL_4 \quad NPL_{\ldots} \\
& \quad \downarrow \quad \downarrow \quad \downarrow \\
L' & \quad L
\end{align*}
\]
Finding an $L'$

- The key property that every NP language has is that it can be decided by a polynomial time NDTM.

- So we need a language in which we can describe computations of NDTMs.
The Cook-Levin Theorem

Define: SAT = \{w : w is a wff in Boolean logic and w is satisfiable\}

Theorem: SAT is NP-complete.

Proof:

- SAT is in NP.
- SAT is NP-hard.
SAT is NP-Hard

• Let $L$ be any language in NP.
• Let $M$ be one of the NDTMs that decides $L$.

Define an algorithm that, given $M$, constructs a reduction $R$ with the property that:

$$w \in L \iff R(w) \in \text{SAT}.$$ 

$R$ takes a string $w$ and returns a Boolean wff that is satisfiable iff $w \in L$. 
SAT is NP-Hard

We need a formula that guarantees that:

1. The table is legal.
2. The first row describes the start of $M$ on $w$.
3. Some row describes an accepting configuration.
4. Each row follows from the previous one.
SAT is NP-Hard

Guarantee that the table is legal:

\[\text{Conjunct}_1 \equiv \text{Tapes} \land \text{States}\]

\[
\begin{align*}
\text{Tapes} & \equiv \bigwedge_{1 \leq i \leq \text{rows}} \left( \bigwedge_{1 \leq j \leq \text{cols}} T_{i,j} \right) \\
T_{i,j} & \equiv \bigvee_{c \in \Gamma} \left( \text{tape}_{i,j,c} \land \left( \bigwedge_{s \neq c} \neg \text{tape}_{i,j,s} \right) \right) \\
\text{States} & \equiv \bigwedge_{1 \leq i \leq \text{rows}} \left( \bigvee_{1 \leq j \leq \text{cols}} \left( Q_{i,j} \land \left( \bigwedge_{k \neq j} \left( \bigwedge_{q \in K} \neg \text{state}_{i,k,q} \right) \right) \right) \right) \\
Q_{i,j} & \equiv \bigvee_{q \in K} \left( \text{state}_{i,j,q} \land \left( \bigwedge_{p \neq q} \neg \text{state}_{i,j,p} \right) \right)
\end{align*}
\]
SAT is NP-Hard

\[
\begin{array}{cccccccc}
\hline
& f(|w|) & & \hline
\hline
& q_0 & a & a & b & \square & \square & \square & \square \\
\hline
& q_1 & a & a & b & \square & \square & \square & \square \\
\hline
& a & q_1 & a & b & \square & \square & \square & \square \\
\hline
& a & b & q_1 & b & \square & \square & \square & \square \\
\hline
& a & b & b & q_2 & \square & \square & \square & \square \\
\hline
& a & b & b & \square & y & \square & \square & \square \\
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\hline
\text{max}(f(|w|), |w|) & & \hline
\hline
a & a & b & \square & \square & \square & \square & \square \\
\hline
a & a & b & \square & \square & \square & \square & \square \\
\hline
a & a & b & \square & \square & \square & \square & \square \\
\hline
a & a & b & \square & \square & \square & \square & \square \\
\hline
\end{array}
\]

Guarantee that row 1 describes the start:

\[
\text{Conjunct}_2 \equiv \text{Blanks} \land \text{Initial}w \land \text{Initial}q
\]

\[
\text{Blanks} \equiv (\bigwedge_{1 \leq j \leq \text{padleft} + 1} \text{tape}_{1,j,\text{blank}}) \land (\bigwedge_{\text{padleft} + 2 \leq j \leq \text{padleft} + |w| + 1} \text{tape}_{1,j,\text{blank}})
\]

\[
\text{Initial}w \equiv \bigwedge_{\text{padleft} + 2 \leq j \leq \text{padleft} + |w| + 1} \text{tape}_{1,j,wj}
\]

\[
\text{Initial}q \equiv \text{state}_{1,\text{padleft} + 1,q0}
\]
SAT is NP-Hard

\[ f(|w|) \quad \text{and} \quad \max(f(|w|), |w|) \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>( q_0 )</th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

\[ f(|w|)+1 \]

\[
\text{Guarantee that some row contains the state } y: \\
Conjunct_3 \equiv \bigvee_{1 \leq i \leq \text{rows}} \bigvee_{1 \leq j \leq \text{cols}} state_{i,j,y}
\]
SAT is NP-Hard

Guarantee that each row follows from the previous one:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1/a$</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>$q_1/a$</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>$q_1/a$</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b</td>
<td>$q_1/b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>$q_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Most Squares Didn’t Change

All the tape squares that aren’t under the read/write head stayed the same. Let:

$$Sames = \forall 2 \leq i \leq \text{done}$$

$$\forall j \ (\forall c \ (\text{read/write head not in column } j \text{ in row } i \rightarrow$$

$$(\text{tape}_{i,j,c} \leftrightarrow \text{tape}_{i-1,j,c})))$$.
The Square Under the Read Head

The tape square under the read/write head changed in some way that is allowed by $\Delta$:

$\text{ChangedTape} \equiv$

$\forall 2 \leq i \leq \text{done} \ (\forall j \ (\forall c \ (\text{read/write head in column } j \text{ in row } i-1 \text{ and } \text{tape}_{i,j,c} \rightarrow \exists p \ (\text{state stored in row } i-1 = p, \text{ and } \exists s \ (\text{character in column } j \text{ in row } i-1 = s, \text{ and } \exists q \ (((p, s), (q, c, (\rightarrow|\leftarrow))) \in \Delta ))))))$. 
Δ Determines Changes

The state and the read/write head changed in some way that is allowed by Δ.

There are two possibilities:
• the read/write head moved one square to the right, or
• it moved one square to the left:

\[ \text{ChangedStateAndHead} \equiv \forall 2 \leq i \leq \text{done} \ (\forall j \ (\forall q \ (\text{state}_{i,j}, q \rightarrow \text{moved-right} \lor \text{moved-left}))) , \]

\[ \text{moved-right} \equiv (\exists p \ (\text{state stored in row } i-1 = p, \text{ and} \]
\[ \exists s \ (\text{character in column } j-1 \text{ in row } i-1 = s, \text{ and} \]
\[ \exists c \ (((p, s), (q, c, \rightarrow)) \in \Delta ))) . \]

\[ \text{moved-left} \equiv (\exists p \ (\text{state stored in row } i-1 = p, \text{ and} \]
\[ \exists s \ (\text{character in column } j+1 \text{ in row } i-1 = s, \text{ and} \]
\[ \exists c \ (((p, s), (q, c, \leftarrow)) \in \Delta ))) . \]
SAT is NP-Hard

On input $w$, $R$ uses $\langle M \rangle$ and constructs a description of the Boolean formula:

$$\text{DescribeMonw} = \text{Conj}_1 \land \text{Conj}_2 \land \text{Conj}_3 \land \text{Conj}_4.$$  

$\text{DescribeMonw}$ will have a satisfying assignment to its variables iff there exists some computational path along which $M$ accepts $w$.

So, for any NP language $L$, $L \leq SAT$.

It remains to show that $R(w)$ operates in polynomial time.
NP-Complete Languages

- **SUBSET-SUM** = \{<S, k> : S is a multiset of integers, k is an integer, and there exists some subset of S whose elements sum to k\}.

- **SET-PARTITION** = \{<S> : S is a multiset of objects each of which has an associated cost and there exists a way to divide S into two subsets, A and S - A, such that the sum of the costs of the elements in A equals the sum of the costs of the elements in S - A\}.

- **KNAPSACK** = \{<S, v, c> : S is a set of objects each of which has an associated cost and an associated value, v and c are integers, and there exists some way of choosing elements of S (duplicates allowed) such that the total cost of the chosen objects is at most c and their total value is at least v\}.
NP-Complete Languages

• TSP-DECIDE.

• HAMILTONIAN-PATH = \{<G> : G is an undirected graph and G contains a Hamiltonian path\}.

• HAMILTONIAN-CIRCUIT = \{<G> : G is an undirected graph and G contains a Hamiltonian circuit\}.

• CLIQUE = \{<G, k> : G is an undirected graph with vertices V and edges E, k is an integer, 1 \leq k \leq |V|, and G contains a k-clique\}.

• INDEPENDENT-SET = \{<G, k> : G is an undirected graph and G contains an independent set of at least k vertices\}.
NP-Complete Languages

• SUBGRAPH-ISOMORPHISM = \{<G_1, G_2> : G_1 \text{ is isomorphic to some subgraph of } G_2\}.

Two graphs $G$ and $H$ are isomorphic to each other iff there exists a way to rename the vertices of $G$ so that the result is equal to $H$. Another way to think about isomorphism is that two graphs are isomorphic iff their drawings are identical except for the labels on the vertices.
SUBGRAPH-ISOMORPHISM

PROPANAL

PROPENAL
NP-Complete Languages

- BIN-PACKING = \{<S, c, k> : S is a set of objects each of which has an associated size and it is possible to divide the objects so that they fit into k bins, each of which has size c}\.
BIN-PACKING

In two dimensions:

Source: mainlinemedia.com
BIN-PACKING

In three dimensions:
NP-Complete Languages

- **SHORTEST-SUPERSTRING** = \{<S, k> : S is a set of strings and there exists some superstring T such that every element of S is a substring of T and T has length less than or equal to k\}. 
SHORTEST-SUPERSTRING

Fig 2: Short fragments of DNA sequence are ordered by overlapping data to recreate the whole genome sequence

Source: Wiley: Interactive Concepts in Biology
NP-Complete Languages

- **BOUNDDED-PCP** = \{<P, k> : P is an instance of the Post Correspondence problem that has a solution of length less than or equal to k\}.

<table>
<thead>
<tr>
<th>i</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>aab</td>
</tr>
<tr>
<td>2</td>
<td>abb</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>aba</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>baaa</td>
<td>baba</td>
</tr>
</tbody>
</table>
Proving that $L$ is NP-Complete

Theorem:

If:

$L_1$ is NP-complete,
$L_1 \leq_P L_2$, and
$L_2$ is in NP,

Then $L_2$ is also NP-complete.
Proving that L is NP-Complete

**Theorem:** If \( L_1 \) is NP-complete, \( L_1 \leq_P L_2 \), and \( L_2 \) is in NP, then \( L_2 \) is also NP-complete.

**Proof:** If \( L_1 \) is NP-complete then every other NP language is deterministic, polynomial-time reducible to it. So let \( L \) be any NP language and let \( R_L \) be the Turing machine that reduces \( L \) to \( L_1 \). If \( L_1 \leq_P L_2 \), let \( R_2 \) be the Turing machine that implements that reduction. Then \( L \) can be deterministic, polynomial-time reduced to \( L_2 \) by first applying \( R_L \) and then applying \( R_2 \). Since \( L_2 \) is in NP and every other language in NP is deterministic, polynomial-time reducible to it, it is NP-complete.
3-SAT

**Define:** 3-SAT = \{<w> : w is a wff in Boolean logic, w is in 3-conjunctive normal form and w is satisfiable\}.

\[(P \lor R \lor \neg T) \land (S \lor \neg R \lor W)\]

**Theorem:** 3-SAT is NP-complete.

**Proof:** We have shown that 3-SAT is in NP.

What about NP-hard?
3-SAT

First we try a reduction from SAT:

\[ R(w: \text{ wff of Boolean logic}) = \]

1. Use \textit{conjunctiveBoolean} to construct \( w' \), where \( w' \) is in conjunctive normal form and \( w' \) is equivalent to \( w \).

2. Use \textit{3-conjunctiveBoolean} to construct \( w'' \), where \( w'' \) is in 3-conjunctive normal form and \( w'' \) is satisfiable iff \( w' \) is.

3. Return \( w'' \).

Does \( R \) run in polynomial time?
Converting to CNF

\[(p \land q) \lor (r \land s) \lor (t \land v) \lor (w \land x)\]

\[((p \lor r) \land (q \lor r) \land (p \lor s) \land (q \lor s)) \lor \]
3-SAT is NP-Hard

Idea 1: Retain the idea of reducing SAT to 3-SAT.

For $R$ to be a reduction from SAT to 3-SAT, it is sufficient to assure that $w'$ is satisfiable iff $w$ is.

There exists a polynomial-time algorithm that constructs, from any wff $w$, a $w'$ that meets that requirement.

If we replace step one of $R$ with that algorithm, $R$ is a polynomial-time reduction from SAT to 3-SAT.

So 3-SAT is NP-hard.
3-SAT is NP-Hard

Idea 2: Prove that 3-SAT is NP-hard directly.

It is possible to modify the reduction $R$ that proves the Cook-Levin Theorem so that it constructs a formula in conjunctive normal form. $R$ will still run in polynomial time.

Once $R$ has constructed a conjunctive normal form formula $w$, we can use $3$-conjunctiveBoolean to construct $w'$, where $w'$ is in 3-conjunctive normal form and $w'$ is satisfiable iff $w$ is.

This composition of $3$-conjunctiveBoolean with $R$ shows that any NP language can be reduced to 3-SAT.

So 3-SAT is NP-hard.
INDEPENDENT-SET

INDEPENDENT-SET = {<G, k> : G is an undirected graph and G contains an independent set of at least k vertices}. 
**Theorem:** INDEPENDENT-SET is NP-complete.

**Proof:**

- INDEPENDENT-SET is NP-hard because:
  
  \[ 3\text{-SAT} \leq_p \text{INDEPENDENT-SET}. \]

- INDEPENDENT-SET is in NP:
INDEPENDENT-SET is in NP

Proof: $Ver(<G, k, c>) =$

1. Check that the number of vertices in $c$ is at least $k$ and no more than $|V|$. If it is not, reject.

2. For each vertex $v$ in $c$:
   
   For each edge $e$ in $E$ that has $v$ as one endpoint:
   
   Check that the other endpoint of $e$ is not in $c$.

$Time_{req}(Ver) \in O(|c| \cdot |E| \cdot |c|)$.

$|c|$ and $|E|$ are polynomial in $|<G, k>|$.

So $Ver$ runs in polynomial time.
NP-Complete Problems So Far

SAT

3-SAT

INDEPENDENT-SET
VERTEX-COVER

- VERTEX-COVER = \{<G, k>: G is an undirected graph and there exists a vertex cover of G that contains at most k vertices}\.

A **vertex cover** $C$ of a graph $G = (V, E)$ is a subset of $V$ such that every edge in $E$ touches at least one of the vertices in $C$.

To be able to test every link in a network, it suffices to place monitors at a set of vertices that form a vertex cover of the network.
VERTEX-COVER

**Theorem:** VERTEX-COVER is NP-complete.

**Proof:** We must prove:

- VERTEX-COVER is in NP, and
- VERTEX-COVER is NP-hard.
VERTEX-COVER is in NP

Proof: \(\text{Ver}(<G, k, c>) = \)

1. Check that the number of vertices in \(c\) is at most \(\min(k, |V|)\). If not, reject.
2. For each vertex \(v\) in \(c\) do:
   - Find all edges in \(E\) that have \(v\) as one endpoint
   - and mark each such edge.
3. Make one final pass through \(E\) and check whether every edge is marked. If all of them are, accept; otherwise reject.

\(\text{Timereq}(\text{Ver}) \in \mathcal{O}(|c| \cdot |E|).\)

Both \(|c|\) and \(|E|\) are polynomial in \(|<G, k>|\). So \(\text{Ver}\) runs in polynomial time.
VERTEX-COVER is NP-Hard

Proof: By reduction from 3-SAT:

Given a wff \( f \), \( R \) will exploit two kinds of gadgets:

• A variable gadget: For each variable \( x \) in \( f \), \( R \) will build a simple graph with two vertices and one edge between them. Label one of the vertices \( x \) and the other one \( \neg x \).

• A clause gadget: For each clause \( c \) in \( f \), \( R \) will build a graph with three vertices, one for each literal in \( c \). There will be an edge between each pair of vertices in this graph.

Then \( R \) will build an edge from every vertex in a clause gadget to the vertex of the variable gadget with the same label.
VERTEX-COVER

\((P \lor \neg Q \lor T) \land (\neg P \lor Q \lor S)\)
VERTEX-COVER

\[ R(<f>) = \]
1. Build a graph \( G \) as described above.
2. Let \( k = v + 2c \).
3. Return \( <G, k> \).

\( R \) runs in polynomial time. To show that it is correct, we must show that:

\[ <f> \in 3\text{-SAT} \iff R(<f>) \in \text{VERTEX-COVER}. \]
**VERTEX-COVER**

\(<f> \in 3\text{-SAT} \rightarrow R(<f>) \in \text{VERTEX-COVER}: \) There exists a satisfying assignment \(A\) for \(f\). \(G\) contains a vertex cover \(C\) of size \(k\):

1. From each variable gadget, add to \(C\) the vertex that corresponds to the literal that is true in \(A\).
2. Since \(A\) is a satisfying assignment, there must exist at least one true literal in each clause. Pick one and put the vertices corresponding to the other two into \(C\).

\(C\) contains exactly \(k\) vertices. And it is a cover of \(G\):

![Graph](image)
\textbf{VERTEX-COVER}

\( R(<f>) \in \text{VERTEX-COVER} \rightarrow <f> \in \text{3-SAT} \): The graph \( G \) that \( R \) builds contains a vertex cover \( C \) of size \( k \). \( C \) must:

- Contain at least one vertex from each variable gadget in order to cover the internal edge in the variable gadget.
- Contain at least two vertices from each clause gadget in order to cover all three internal edges in the clause gadget.

Satisfying those two requirements uses up all \( k = v + 2c \) vertices, so the vertices we have just described are the only vertices in \( C \).
We can use $C$ to show that there exists some satisfying assignment $A$ for $f$.

To build $A$, assign the value $True$ to each literal that is the label for one of the vertices of $C$ that comes from a variable gadget.

$A$ is a satisfying assignment for $f$ iff it assigns the value $True$ to at least one literal in each of $f$'s clauses. $A$ must do that:
TSP-DECIDE

All of these languages are NP-complete:

3-SAT
\[ \geq_p \]
DIRECTED-HAMILTONIAN-CIRCUIT
\[ \geq_p \]
HAMILTONIAN-CIRCUIT
\[ \geq_p \]
TSP-DECIDE
DIRECTED-HAMILTONIAN-CIRCUIT in NP

Proof: \( \text{Ver}(<G, c>) = \)

1. Reject if either:
   - The number of vertices in \( c \) is not \( |V|+1 \), or
   - The first and last vertices in \( c \) are not identical.
2. For each vertex \( v \) in \( c \), except the last, do:
   2.1. Mark \( v \) in \( V \) and reject if it had previously been marked.
   2.2. Check that the required edge to \( v \) exists and reject if it does not.
3. All tests have passed: accept.

\( \text{Time}_{\text{req}}(\text{Ver}) \in \mathcal{O}(|c| \cdot (|V| + |E|)). \)

All of \( |c|, |V|, \) and \( |E| \) are polynomial in \( |<G, k>| \). So \( \text{Ver} \) runs in polynomial time.
DIRECTED-HAMILTONIAN-CIRCUIT is NP-Hard

Proof: By a reduction $R$ from 3-SAT:
DIRECTED-HAMILTONIAN-CIRCUIT

The variable gadgets: For Boolean formula \( Bf \) with \( n \) variables:

If \( v \) is the \( i^{th} \) such variable, let \( m \) be the larger of the number of occurrences of \( v \) or of \( \neg v \) in \( Bf \). Then \( V_i \) is:

\[ \begin{align*}
   & a_i \quad f_{i,1} \quad f_{i,2} \quad f_{i,2} \\
   & t_{i,1} \quad t_{i,2} \quad t_{i,2} \\
   & \quad \quad \ldots \quad \quad \quad \\
   & f_{i,m} \quad t_{i,m} \quad b_i
\end{align*} \]
There are two Hamiltonian paths through $V_i$:

- The one that begins by going down to a $t$ vertex. We will use this one to correspond to assigning to the variable $v$ the value True.
- The one that begins by going up to an $f$ vertex. We will use this one to correspond to assigning to the variable $v$ the value False.
DIRECTED-HAMILTONIAN-CIRCUIT

Combining the variable gadgets:
DIRECTED-HAMILTONIAN-CIRCUIT

A clause gadget:
DIRECTED-HAMILTONIAN-CIRCUIT

Combining the variable and clause gadgets
The reduction is correct:
HAMILTONIAN-CIRCUIT

**Theorem:** HAMILTONIAN-CIRCUIT is in NP.

**Proof:**

**Theorem:** HAMILTONIAN-CIRCUIT is NP-hard:

**Proof:** By reduction from:

DIRECTED-HAMILTONIAN-CIRCUIT.
HAMILTONIAN-CIRCUIT is NP-Hard

Given a directed graph $G$, $R$ will build an undirected graph $G'$:

Each of $G$’s vertices will be represented in $G'$ by a gadget that contains three vertices connected by two edges.

If there is a directed edge in $G$ from $v$ to $w$, then $G'$ will contain an (undirected) edge from the last of the vertices in $v$’s gadget to the first of the vertices in $w$’s gadget.
HAMILTONIAN-CIRCUIT

G contains a Hamiltonian circuit.

R builds:
HAMILTONIAN-CIRCUIT

G does not contain a Hamiltonian circuit.

R builds:
HAMILTONIAN-CIRCUIT

(<G> ∈ DIRECTED-HAMILTONIAN-CIRCUIT) →
(R(<G>) ∈ HAMILTONIAN-CIRCUIT):

G contains some Hamiltonian circuit
H = (v_1, v_2, ..., v_k, v_1).

Then a Hamiltonian circuit through
R(<G>) is:

(v_{1a}, v_{1b}, v_{1c}, v_{2a}, ..., v_{ka}, v_{kb}, v_{kc}, v_{1a})
Any Hamiltonian circuit $H$ must move in the same direction through all vertex gadgets. If it traverses the vertex gadgets $(v_1, v_2, ..., v_k, v_1)$, in that order, then $(v_1, v_2, ..., v_k, v_1)$ is a Hamiltonian circuit through $G$. 
TSP-DECIDE

Theorem: TSP-DECIDE is in NP.

Proof: Already done.

Theorem: TSP-DECIDE is NP-hard:

Proof: By reduction from:

HAMiltonian-Circuit.
TSP-DECIDE is NP-Hard

Let $G = (V, E)$ be an unweighted, undirected graph. If there is a Hamiltonian circuit through $G$, it must contain exactly $|V|$ edges. So $R$ is:

$$R(<G>) =$$
1. From $G$ construct $G'$, identical to $G$ except that each edge will be assigned the cost 1.

$R$ runs in polynomial time. And it is correct since $G$ has a Hamiltonian circuit iff $G'$ has one with cost equal to $|V|$. 