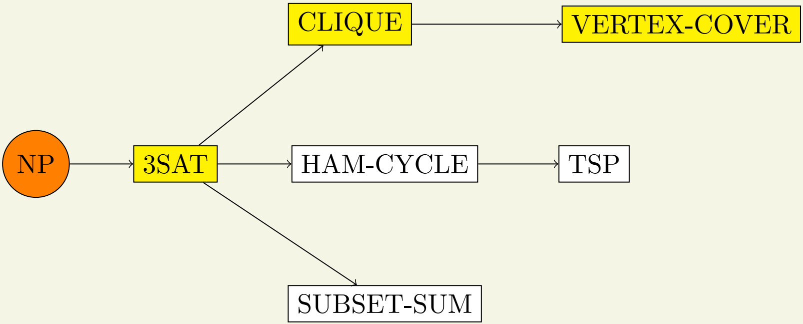
Computability and Complexity
COSC 4200

NP-Complete Problems II

Plan for NP-completeness reductions:



Hamiltonian Cycle

Recall that a Hamiltonian cycle in a graph is a cycle that visits all the vertices without repetition. Not all graphs have a Hamiltonian cycle, and determining if a graph has such a cycle is NP-complete.

The decision problem is

$$\text{HAM-CYCLE} = \{ G \,|\, G \text{ has a Hamiltonian cycle} \} .$$

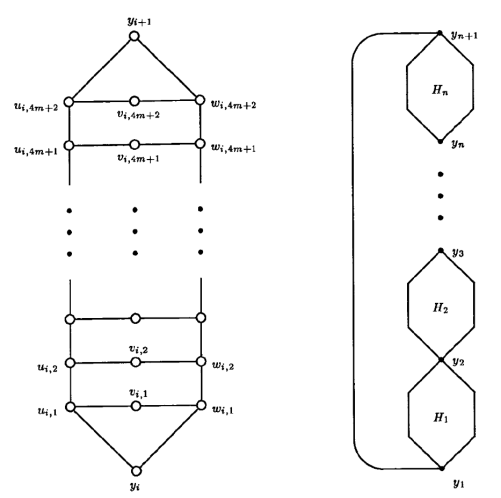
Theorem

$\mathrm{HAM\text{-}CYCLE}$ *is* NP*-complete.*

Proof. We have already seen that $\mathrm{HAM\text{-}CYCLE} \in \mathrm{NP}$. We will do a reduction from $3\mathrm{SAT}$. Let

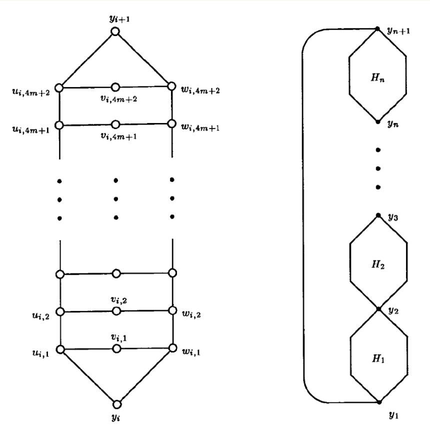
$$\phi = C_1 \wedge \ldots \wedge C_m$$

be a formula over n variables $x_1, \ldots, x_n$.



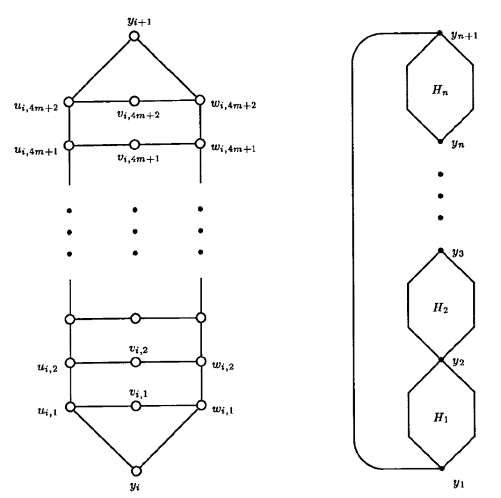
For each x_i, we construct a ladder H_i of $12m+7$ vertices.

- We have $4m+2$ rungs with 3 vertices in each rung, and a vertex at the bottom and at the top.
- The rungs of the ladders are composed of $u_{i,j}$, $v_{i,j}$, and $w_{i,j}$, where i corresponds to H_i, and j corresponds to the rung.
- There is an edge from $u_{i,j}$ to the u above and below it (or to y_i or y_{i+1}) and an edge to $v_{i,j}$. Similarly with $w_{i,j}$.



This is actually $12m + 8$ vertices, but we hook this ladders together at the top and bottom, so we only count, for example, the bottom vertex as belonging to H_i, while the top one belongs to H_{i+1}. The top of the top ladder connects to the bottom of the bottom ladder.

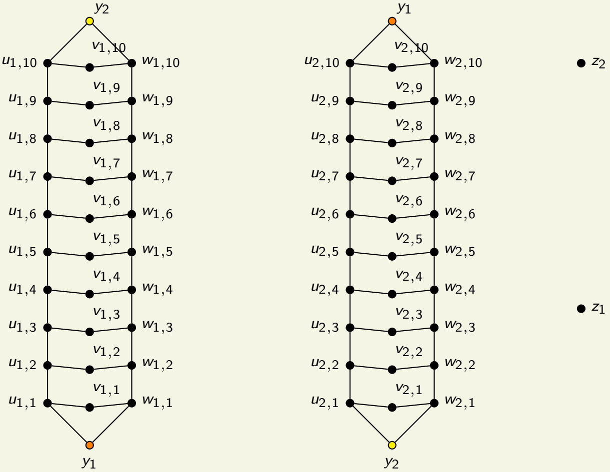
There are two ways to traverse a ladder, either starting by going to the left, or going to the right. Intuitively, going left in H_i corresponds to setting x_i true, and going right corresponds to setting x_i false.



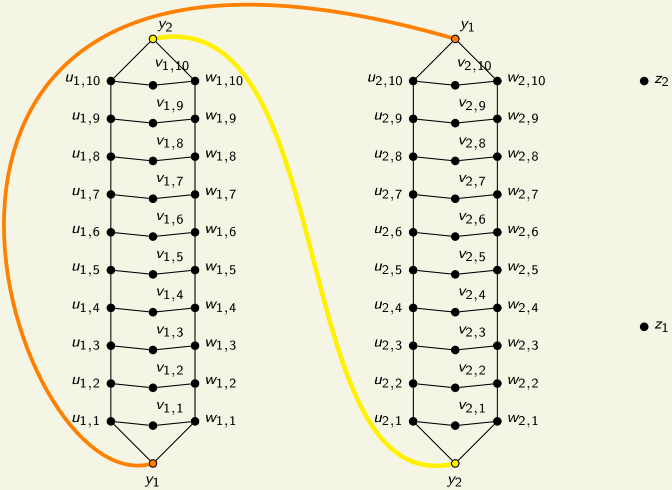
For each clause C_j, we add a new vertex z_j such that

- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.

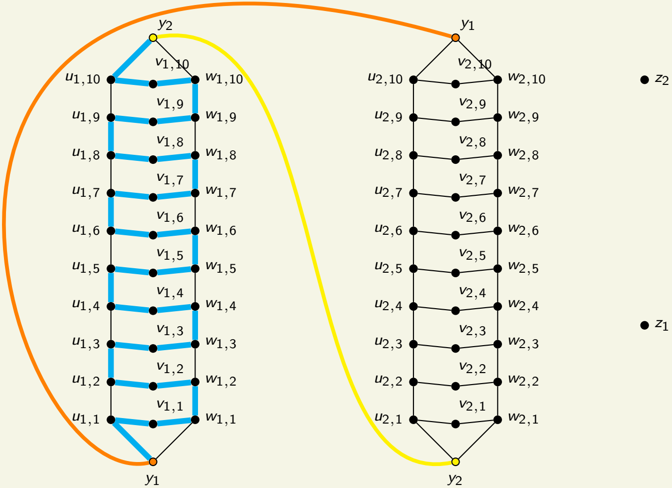
This completes the construction of the graph.
Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.



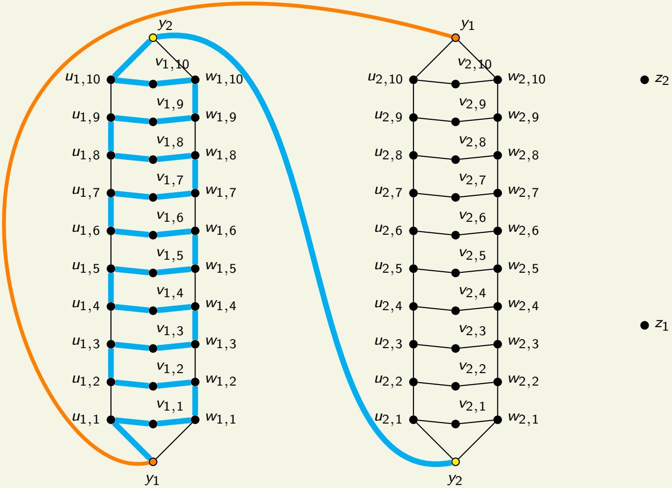
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



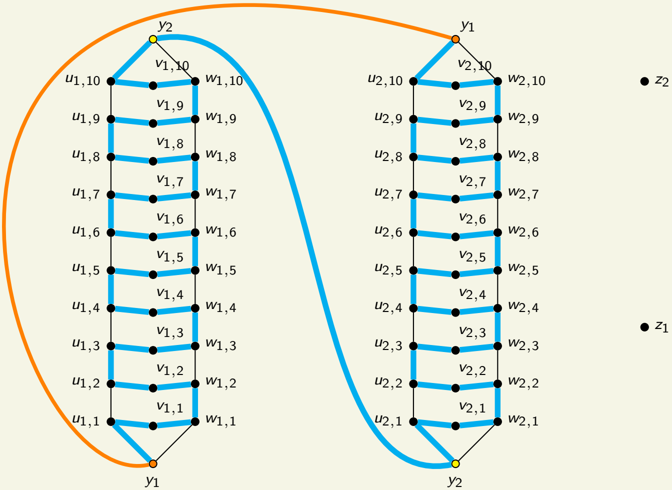
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



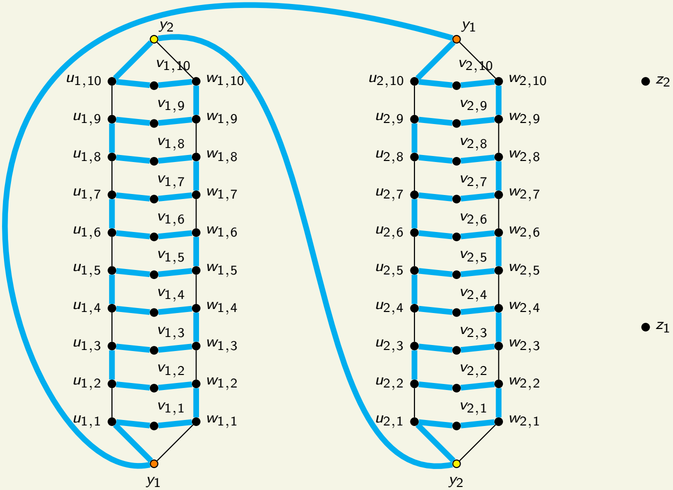
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



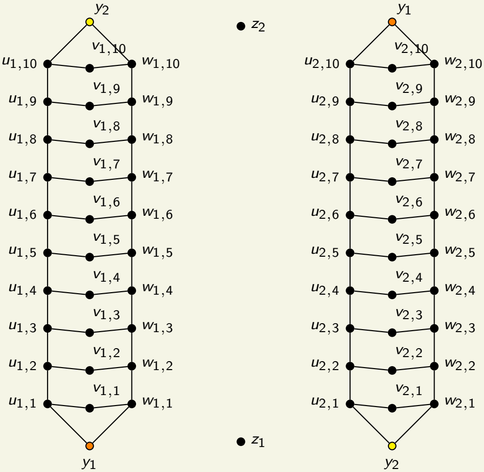
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



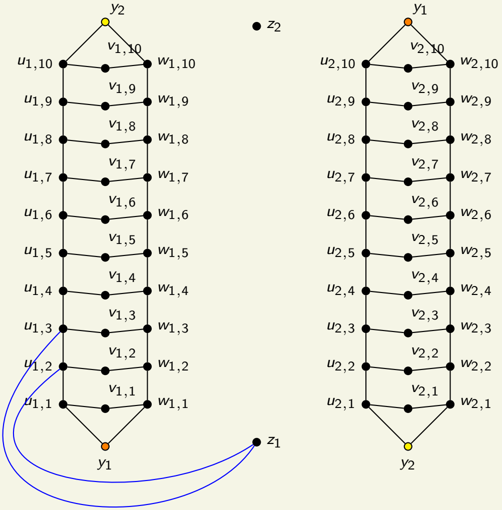
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

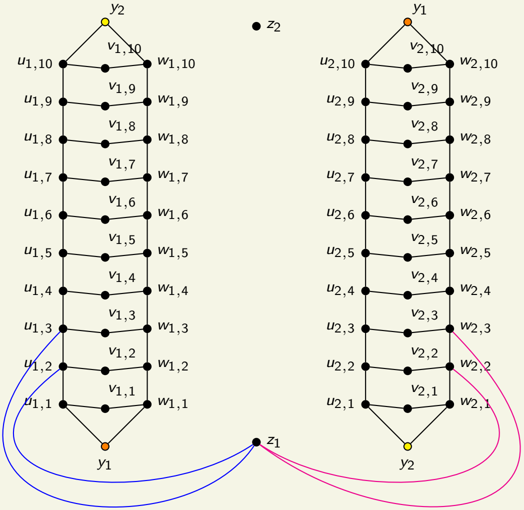
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

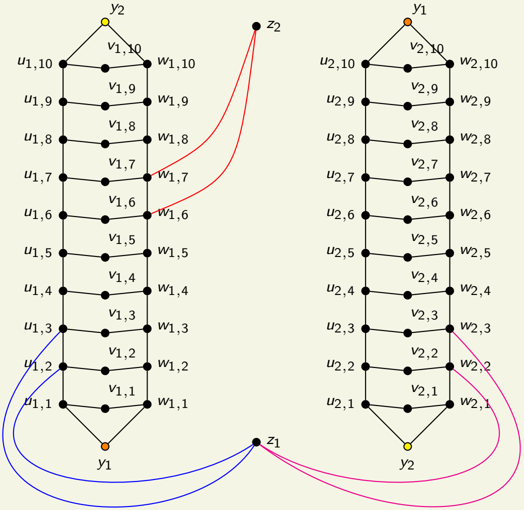
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

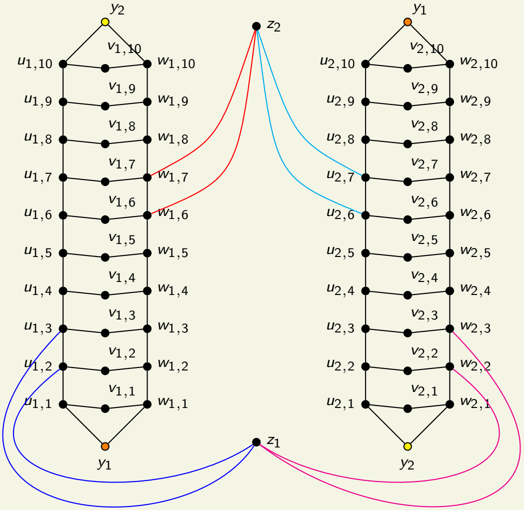
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

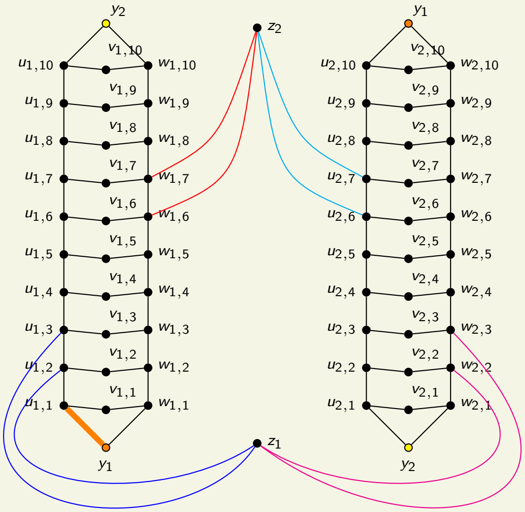
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

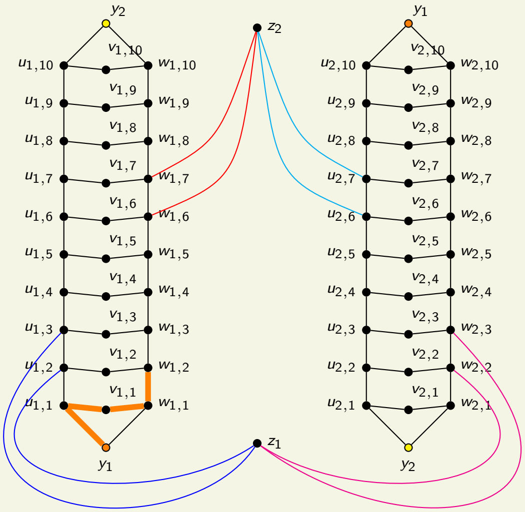
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

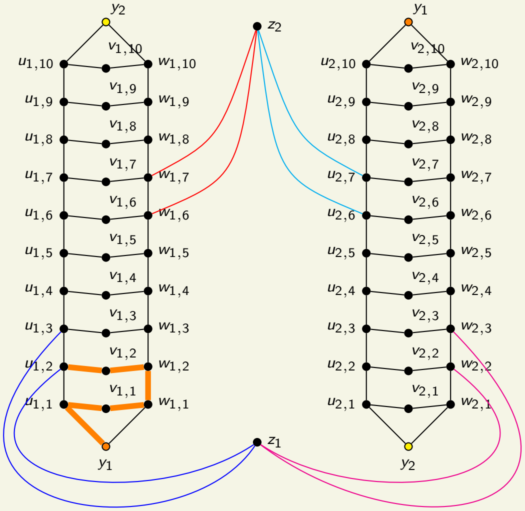
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

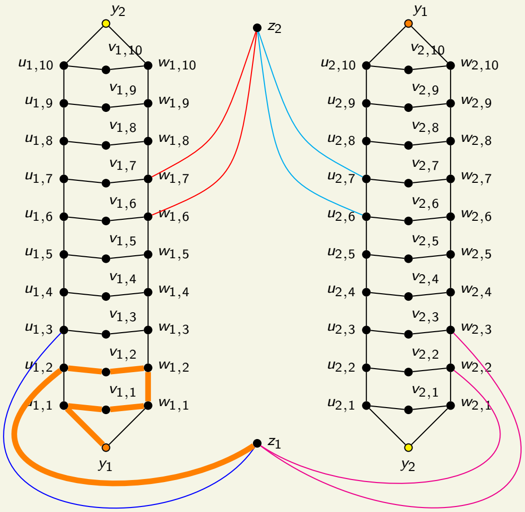
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

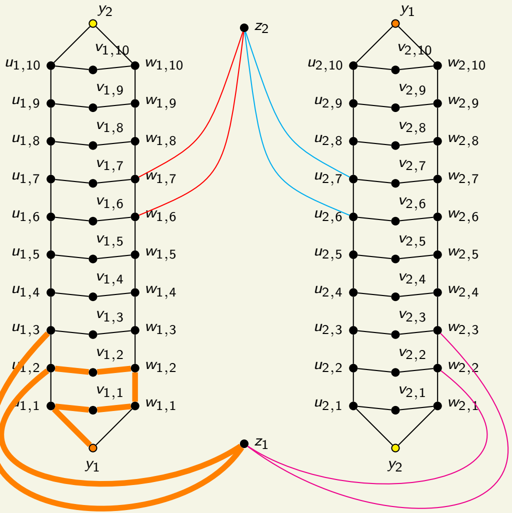
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

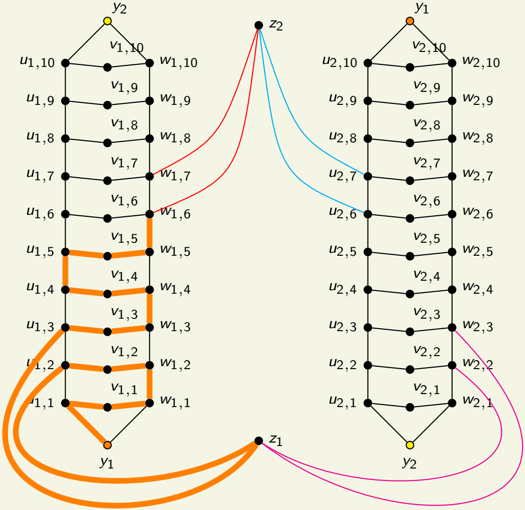
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

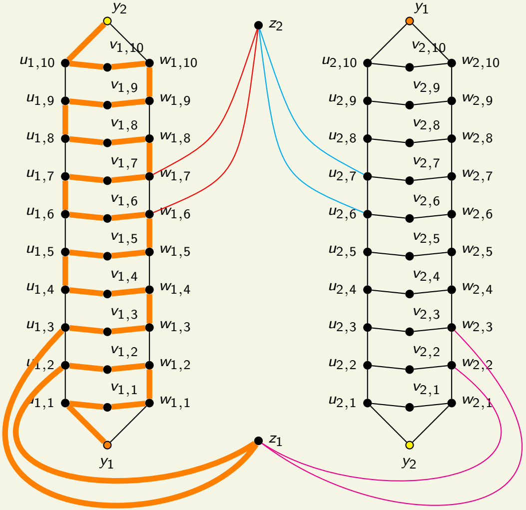
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



For each clause C_j, we add a new vertex z_j such that

- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.

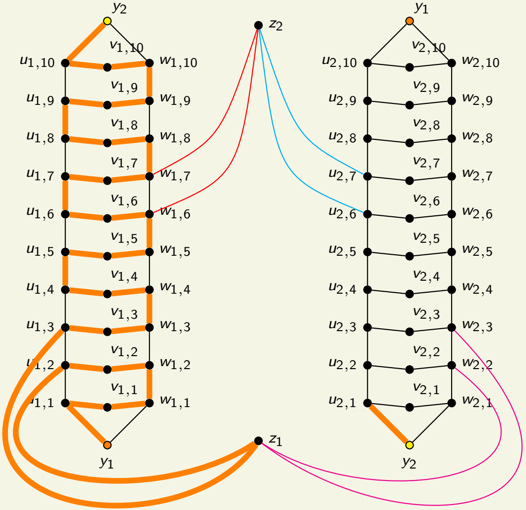
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



For each clause C_j, we add a new vertex z_j such that

- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.

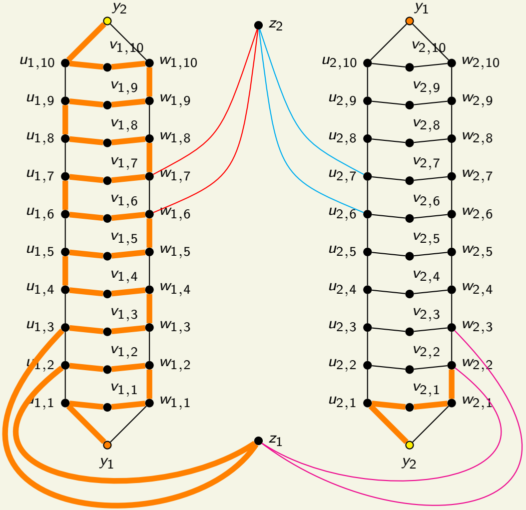
$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

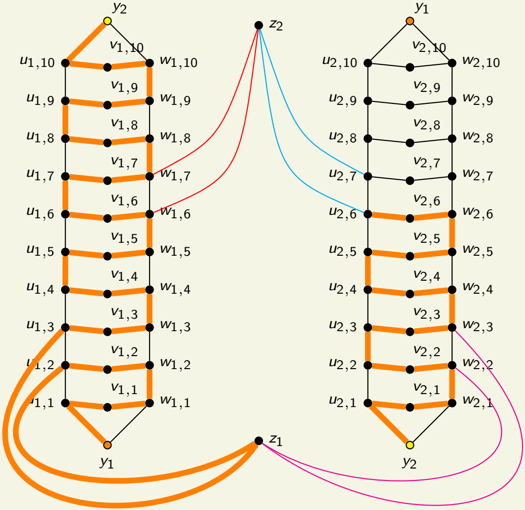
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2)$$

For each clause C_j, we add a new vertex z_j such that

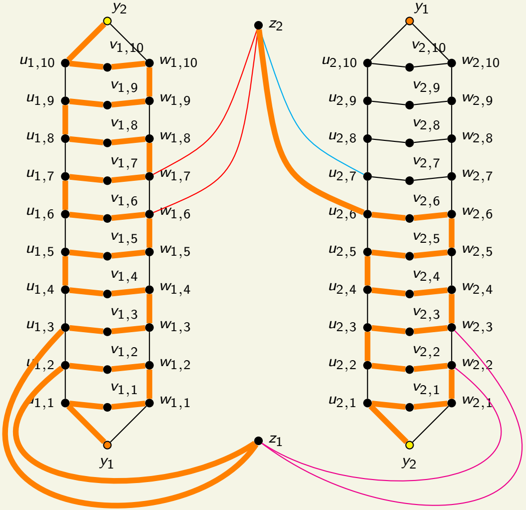
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

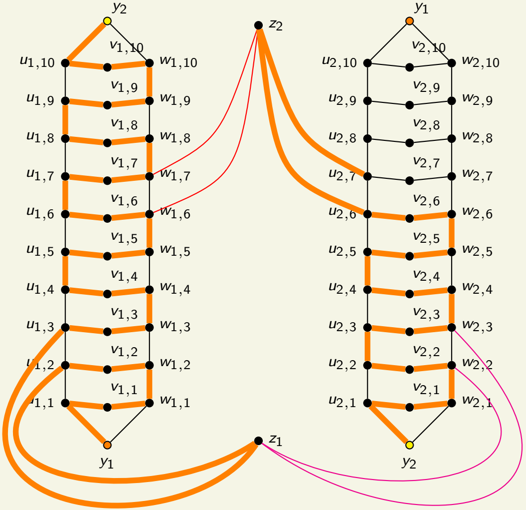
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

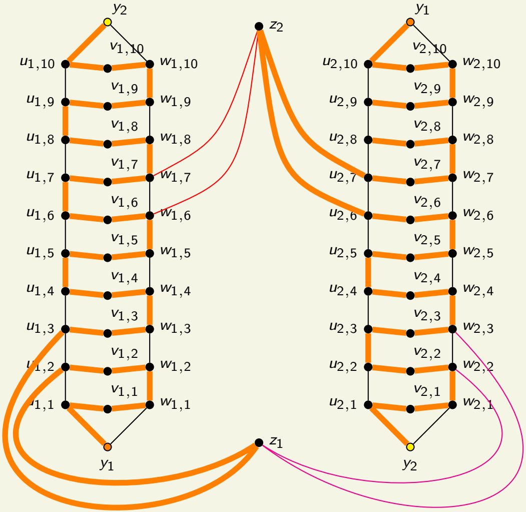
- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.



$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

For each clause C_j, we add a new vertex z_j such that

- If C_j contains the literal x_i, we connect z_j to $u_{i,4j-1}$, and $u_{i,4j-2}$.
- If C_j contains the literal $\neg x_i$, we connect z_j to $w_{i,4j-1}$ and $w_{i,4j-2}$.

Observe that:

- If we go left in a Hamiltonian cycle in ladder H_i, we will traverse
  - the even edges $(u_{i,2}, u_{i,3}), (u_{i,4}, u_{i,5}), \ldots$ on the left side, and
  - the odd edges $(w_{i,1}, w_{i,1}), (w_{i,3}, w_{i,4}), \ldots$ on the right side.

Observe that:

- If we go left in a Hamiltonian cycle in ladder H_i, we will traverse
  - the even edges $(u_{i,2}, u_{i,3}), (u_{i,4}, u_{i,5}), \ldots$ on the left side, and
  - the odd edges $(w_{i,1}, w_{i,1}), (w_{i,3}, w_{i,4}), \ldots$ on the right side.

  If x_i appears as a literal in clause C_j, then z_j will be connected to the endpoints of one of the even edges on the left:

  $$(u_{i,4j-2}, u_{i,4j-1}).$$

  The cycle could safely visit z_j instead of taking this edge. In this way, going left in H_i corresponds to setting $x_i = T$.

- Similarly, if we go right in a Hamiltonian cycle in ladder H_i, we will traverse
  - the odd edges $(u_{i,1}, u_{i,2}), (u_{i,3}, u_{i,4}), \ldots$ on the left side, and
  - the even edges $(w_{i,2}, w_{i,3}), (w_{i,4}, w_{i,5}), \ldots$ on the right side.

- Similarly, if we go right in a Hamiltonian cycle in ladder H_i, we will traverse
  - the odd edges $(u_{i,1}, u_{i,2}), (u_{i,3}, u_{i,4}), \ldots$ on the left side, and
  - the even edges $(w_{i,2}, w_{i,3}), (w_{i,4}, w_{i,5}), \ldots$ on the right side.

  If $\neg x_i$ appears as a literal in clause C_j, then z_j will be connected to the endpoints of one of the even edges on the right:

$$(w_{i,4j-2}, w_{i,4j-1}).$$

  The cycle could safely visit z_j instead of taking this edge. In this way, going left in H_i corresponds to setting $x_i = F$.

Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.

Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.

Suppose ϕ is satisfiable and fix a satisfying assignment. We use this assignment to define a cycle that goes left at H_i if $x_i = T$ and goes right if $x_i = F$.

Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.

Suppose ϕ is satisfiable and fix a satisfying assignment. We use this assignment to define a cycle that goes left at H_i if $x_i = T$ and goes right if $x_i = F$.

- The z_j's are connected to the ladders so that if C_j contains $\neg x_i$, then vertex z_j can be visited by a cycle that goes right in ladder H_i.

Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.

Suppose ϕ is satisfiable and fix a satisfying assignment. We use this assignment to define a cycle that goes left at H_i if $x_i = T$ and goes right if $x_i = F$.

- The z_j's are connected to the ladders so that if C_j contains $\neg x_i$, then vertex z_j can be visited by a cycle that goes right in ladder H_i.
- Similarly, if C_j contains x_i, then vertex z_j can be visited by a cycle that goes left in ladder H_i.

Our claim is that ϕ is satisfiable if and only if the graph has a Hamiltonian cycle.

Suppose ϕ is satisfiable and fix a satisfying assignment. We use this assignment to define a cycle that goes left at H_i if $x_i = T$ and goes right if $x_i = F$.

- The z_j's are connected to the ladders so that if C_j contains $\neg x_i$, then vertex z_j can be visited by a cycle that goes right in ladder H_i.
- Similarly, if C_j contains x_i, then vertex z_j can be visited by a cycle that goes left in ladder H_i.

Thus we can extend the cycle to visit all the z_j's. Therefore if ϕ is satisfiable, then there is a Hamiltonian cycle.

For the other direction, suppose there is a Hamiltonian cycle.

For the other direction, suppose there is a Hamiltonian cycle.

- We set $x_i = T$ if the cycle goes left at H_i and set $x_i = F$ if it goes right.

For the other direction, suppose there is a Hamiltonian cycle.

- We set $x_i = T$ if the cycle goes left at H_i and set $x_i = F$ if it goes right.
- The cycle visits all the z_j's, so it follows from the construction that this assignment satisfies all clauses of ϕ.

Therefore if there is a Hamiltonian cycle, then ϕ is satisfiable.

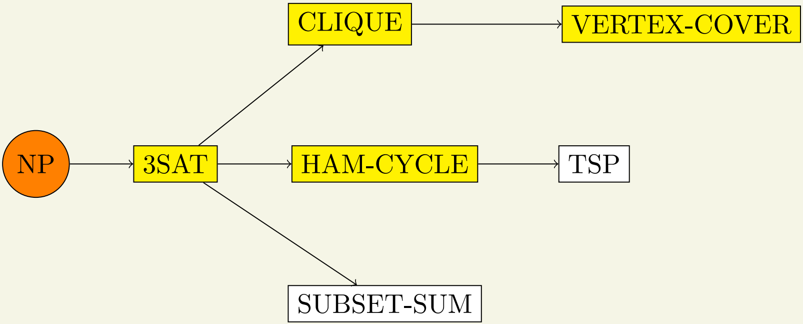
For the other direction, suppose there is a Hamiltonian cycle.

- We set $x_i = T$ if the cycle goes left at H_i and set $x_i = F$ if it goes right.
- The cycle visits all the z_j's, so it follows from the construction that this assignment satisfies all clauses of ϕ.

Therefore if there is a Hamiltonian cycle, then ϕ is satisfiable.

The graph we constructed can computed in quadratic time in the size of the formula. Therefore $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{HAM\text{-}CYCLE}$. $\square$

Plan for NP-completeness reductions:



Traveling Salesman Problem

In the traveling salesman problem, a salesman wishes to visit a collection of cities at the minimum possible cost.

- For each pair of cities, A and B, there is a cost associated with traveling from A to B.
- The goal is to find a tour of the cities of the minimum total cost.

Traveling Salesman Problem

In the traveling salesman problem, a salesman wishes to visit a collection of cities at the minimum possible cost.

- For each pair of cities, A and B, there is a cost associated with traveling from A to B.
- The goal is to find a tour of the cities of the minimum total cost.

We model an instance of this problem as follows:

- n = number of cities. The cities are numbered from 1 to n.
- $c : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \mathbb{N}^+$ is a cost function. $c(i, j)$ is the cost of traveling from city i to city j.

Traveling Salesman Problem

A tour of the n cities is a sequence $(a_1, \ldots, a_n, a_{n+1})$ where

- $a_i \in \{1, \ldots, n\}$ for all $1 \leq i \leq n$,
- $a_{n+1} = a_1$, and
- for each $j \in \{1, \ldots, n\}$, there is some $i \in \{1, \ldots, n\}$ such that $a_i = j$.

Traveling Salesman Problem

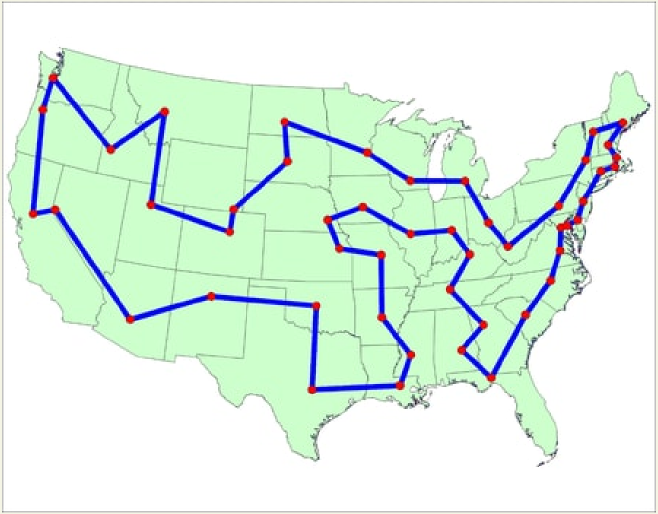
A tour of the n cities is a sequence $(a_1, \ldots, a_n, a_{n+1})$ where

- $a_i \in \{1, \ldots, n\}$ for all $1 \leq i \leq n$,
- $a_{n+1} = a_1$, and
- for each $j \in \{1, \ldots, n\}$, there is some $i \in \{1, \ldots, n\}$ such that $a_i = j$.

The cost of the tour is

$$\sum_{i=1}^{n} c(a_i, a_{i+1})$$

Sometimes this is also called the length of the tour.



A TSP tour of the continental U.S. state capitals.

Traveling Salesman Problem

The traveling salesman decision problem is

$$\mathrm{TSP} = \left\{ \langle n, c, k \rangle \;\middle|\; \begin{array}{l} c \text{ is a cost function and there} \\ \text{is a tour with cost at most } k \end{array} \right\}$$

It is easy to see that $\mathrm{TSP} \in \mathrm{NP}$. The witnesses are the tours. We can easily check if the cost of a tour is at most k.

Theorem

TSP *is* NP*-complete.*

Before we prove this theorem, let's discuss the consequences.

In the TSP decision problem, we only have to decide if a tour with cost at most k exists. In practice what we really want is to find a minimum cost tour.

Theorem

TSP *is* NP*-complete.*

Before we prove this theorem, let's discuss the consequences.

In the TSP decision problem, we only have to decide if a tour with cost at most k exists. In practice what we really want is to find a minimum cost tour. A corollary of the theorem is the following.

Corollary

If $\mathrm{P} \neq \mathrm{NP}$*, then there is no polynomial-time algorithm that finds shortest* TSP *tours.*

Theorem

TSP *is* NP*-complete.*

Before we prove this theorem, let's discuss the consequences.

In the TSP decision problem, we only have to decide if a tour with cost at most k exists. In practice what we really want is to find a minimum cost tour. A corollary of the theorem is the following.

Corollary

If $\mathrm{P} \neq \mathrm{NP}$*, then there is no polynomial-time algorithm that finds shortest* TSP *tours.*

Proof. Suppose we have a polynomial-time algorithm for finding the shortest tours. Then given an instance $\langle n, c, k \rangle$ of TSP, we can compute the shortest tour and see if its cost is at most k. This shows that $\mathrm{TSP} \in \mathrm{P}$, so $\mathrm{P} = \mathrm{NP}$ because TSP is NP-complete. $\square$

Theorem

TSP *is* NP*-complete.*

Proof. We will reduce HAM-CYCLE to TSP. Let $G = (V, E)$ be a graph.

Theorem

TSP *is* NP*-complete.*

Proof. We will reduce $\mathrm{HAM\text{-}CYCLE}$ to TSP. Let $G = (V, E)$ be a graph.

Define $n = |V|$, $k = n$, and

$$c(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ n+1 & \text{if } (i,j) \notin E \end{cases}$$

for all $i, j \in \{1, \ldots, n\}$.

Theorem

TSP *is* NP*-complete.*

Proof. We will reduce $\mathrm{HAM\text{-}CYCLE}$ to TSP. Let $G = (V, E)$ be a graph.

Define $n = |V|$, $k = n$, and

$$c(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ n+1 & \text{if } (i,j) \notin E \end{cases}$$

for all $i, j \in \{1, \ldots, n\}$.

We claim that

$$G \in \mathrm{HAM\text{-}CYCLE} \iff \langle n, c, k \rangle \in \mathrm{TSP}.$$

Suppose that G has a Hamiltonian cycle. Then that cycle is a tour of length $n = k$ in the TSP instance. Therefore $G \in \mathrm{HAM\text{-}CYCLE}$ implies $\langle n, c, k \rangle \in \mathrm{TSP}$.

Suppose that G has a Hamiltonian cycle. Then that cycle is a tour of length $n = k$ in the TSP instance. Therefore $G \in \mathrm{HAM\text{-}CYCLE}$ implies $\langle n, c, k \rangle \in \mathrm{TSP}$.

For the converse, suppose that there is a tour with cost $\leq k = n$. Then that tour never goes from a city i to a city j with a cost of $n + 1$. Therefore, there is an edge in G between every consecutive pair of cities on the tour, so the tour is a Hamiltonian cycle in G. Therefore $\langle n, c, k \rangle \in \mathrm{TSP}$ implies $G \in \mathrm{HAM\text{-}CYCLE}$.

Suppose that G has a Hamiltonian cycle. Then that cycle is a tour of length $n = k$ in the TSP instance. Therefore $G \in \mathrm{HAM\text{-}CYCLE}$ implies $\langle n, c, k \rangle \in \mathrm{TSP}$.

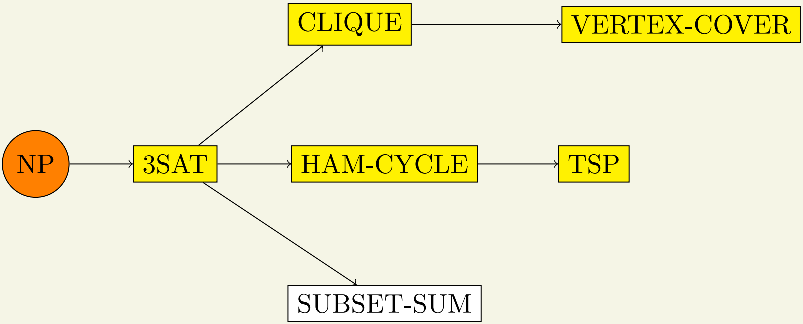
For the converse, suppose that there is a tour with cost $\leq k = n$. Then that tour never goes from a city i to a city j with a cost of $n + 1$. Therefore, there is an edge in G between every consecutive pair of cities on the tour, so the tour is a Hamiltonian cycle in G. Therefore $\langle n, c, k \rangle \in \mathrm{TSP}$ implies $G \in \mathrm{HAM\text{-}CYCLE}$.

Since we can compute $\langle n, c, k \rangle$ from G in polynomial time, we have shown that

$$\mathrm{HAM\text{-}CYCLE} \leq_{\mathrm{P}} \mathrm{TSP}.$$

$\square$

Plan for NP-completeness reductions:



Subset Sum

The subset sum decision problem is

$$\text{SUBSET-SUM} = \left\{ \langle L, S \rangle \;\middle|\; \begin{array}{c} L = (a_1, \ldots, a_n) \text{ is a list of } n \text{ integers} \\ \text{and} \\ \exists I \subseteq \{1, \ldots, n\} \text{ such that } \sum_{i \in I} a_i = S \end{array} \right\}$$

Theorem

SUBSET-SUM *is* NP*-complete.*

Proof. We've seen that $\text{SUBSET-SUM} \in \text{NP}$. We'll reduce 3SAT to SUBSET-SUM.

Let $\phi = C_1 \wedge \ldots \wedge C_m$ be a 3CNF formula over n variables $x_1, \ldots, x_n$.

Theorem

SUBSET-SUM *is* NP*-complete.*

Proof. We've seen that SUBSET-SUM $\in$ NP. We'll reduce 3SAT to SUBSET-SUM.

Let $\phi = C_1 \wedge \ldots \wedge C_m$ be a 3CNF formula over n variables $x_1, \ldots, x_n$.

We will define a list of $2n + 2m$ integers that will all have values between 0 and 10^{n+m}, and a target sum.

We will write them as decimal numbers using exactly $n + m$ digits (using leading 0's when necessary).

For any integer r, let $r[k]$ be the k^{th} most significant digit in the decimal representation of r. So if $r = 0156$, then $r[1] = 0$ and $r[4] = 6$.

For any integer r, let $r[k]$ be the k^{th} most significant digit in the decimal representation of r. So if $r = 0156$, then $r[1] = 0$ and $r[4] = 6$.

We define S, our target sum, by

$$S[k] = \begin{cases} 1 & \text{if } 1 \leq k \leq n \\ 3 & \text{if } n+1 \leq k \leq n+m. \end{cases}$$

In other words, S is the number

$$S = \underbrace{11\ldots1}_{n}\underbrace{33\ldots3}_{m}.$$

For each $1 \leq j \leq n$ we define two numbers $b_{j,0}$ and $b_{j,1}$ as follows.

For each $1 \leq j \leq n$ we define two numbers $b_{j,0}$ and $b_{j,1}$ as follows.

- For all $1 \leq k \leq n$,

$$b_{j,0}[k] = b_{j,1}[k] = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j. \end{cases}$$

For each $1 \leq j \leq n$ we define two numbers $b_{j,0}$ and $b_{j,1}$ as follows.

- For all $1 \leq k \leq n$,

$$b_{j,0}[k] = b_{j,1}[k] = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j. \end{cases}$$

- For all $1 \leq k \leq m$,

$$b_{j,0}[n+k] = \begin{cases} 1 & \text{if } \neg x_j \text{ occurs in } C_k \\ 0 & \text{otherwise.} \end{cases}$$

For each $1 \leq j \leq n$ we define two numbers $b_{j,0}$ and $b_{j,1}$ as follows.

- For all $1 \leq k \leq n$,

$$b_{j,0}[k] = b_{j,1}[k] = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j. \end{cases}$$

- For all $1 \leq k \leq m$,

$$b_{j,0}[n+k] = \begin{cases} 1 & \text{if } \neg x_j \text{ occurs in } C_k \\ 0 & \text{otherwise.} \end{cases}$$

- For all $1 \leq k \leq m$,

$$b_{j,1}[n+k] = \begin{cases} 1 & \text{if } x_j \text{ occurs in } C_k \\ 0 & \text{otherwise.} \end{cases}$$

For each $1 \leq i \leq m$ we define two numbers $c_{i,0}$ and $c_{i,1}$ by

$$c_{i,0}[k] = c_{i,1}[k] = \begin{cases} 1 & \text{if } k = n + i \\ 0 & \text{otherwise} \end{cases}$$

for all $1 \leq k \leq n + m$.

For each $1 \leq i \leq m$ we define two numbers $c_{i,0}$ and $c_{i,1}$ by

$$c_{i,0}[k] = c_{i,1}[k] = \begin{cases} 1 & \text{if } k = n + i \\ 0 & \text{otherwise} \end{cases}$$

for all $1 \leq k \leq n + m$.

Let L be the list of these numbers:

$$L = (b_{1,0}, b_{2,0}, \ldots, b_{n,0}, b_{1,1}, \ldots, b_{n,1}, c_{1,0}, \ldots, c_{n,0}, c_{1,1}, \ldots, c_{n,1})$$

Note that (L, S) can be computed in polynomial time from ϕ.

For each $1 \leq i \leq m$ we define two numbers $c_{i,0}$ and $c_{i,1}$ by

$$c_{i,0}[k] = c_{i,1}[k] = \begin{cases} 1 & \text{if } k = n + i \\ 0 & \text{otherwise} \end{cases}$$

for all $1 \leq k \leq n + m$.

Let L be the list of these numbers:

$$L = (b_{1,0}, b_{2,0}, \ldots, b_{n,0}, b_{1,1}, \ldots, b_{n,1}, c_{1,0}, \ldots, c_{n,0}, c_{1,1}, \ldots, c_{n,1})$$

Note that (L, S) can be computed in polynomial time from ϕ.

Claim

$\phi \in \text{3SAT} \iff (L, S) \in \text{SUBSET-SUM}.$

As an example, consider

$$\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

Then $n = m = 3$, $S = 111333$, and

$$\begin{array}{rcl@{\qquad\qquad}rcl}
b_{1,0} & = & 100101 & c_{1,0} & = & 000100 \\
b_{1,1} & = & 100010 & c_{1,1} & = & 000100 \\
b_{2,0} & = & 010011 & c_{2,0} & = & 000010 \\
b_{2,1} & = & 010100 & c_{2,1} & = & 000010 \\
b_{3,0} & = & 001101 & c_{3,0} & = & 000001 \\
b_{3,1} & = & 001010 & c_{3,1} & = & 000001
\end{array}$$

Observe that for any k, $n < k \leq n + m$, there are at most 5 numbers in L with a 1 in the k^{th} digit. This is because each clause has at most 3 literals, and there are $2+$(the number of literals in the k^{th} clause) many 1's in the k^{th} digits of numbers in L.

Observe that for any k, $n < k \leq n + m$, there are at most 5 numbers in L with a 1 in the k^{th} digit. This is because each clause has at most 3 literals, and there are 2+(the number of literals in the k^{th} clause) many 1's in the k^{th} digits of numbers in L.

Also, for any k, $1 \leq k \leq n$, there are exactly two numbers with a 1 in the k^{th} digit, namely $b_{k,0}$ and $b_{k,1}$.

Observe that for any k, $n < k \leq n + m$, there are at most 5 numbers in L with a 1 in the k^{th} digit. This is because each clause has at most 3 literals, and there are 2+(the number of literals in the k^{th} clause) many 1's in the k^{th} digits of numbers in L.

Also, for any k, $1 \leq k \leq n$, there are exactly two numbers with a 1 in the k^{th} digit, namely $b_{k,0}$ and $b_{k,1}$.

Therefore to get a sum of S we need to choose a subset of the numbers that

- has exactly three numbers with a 1 in the k^{th} position for $n < k \leq n + m$ and
- has either $b_{k,0}$ or $b_{k,1}$ for $1 \leq k \leq n$.

$$\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

$$\begin{array}{rclcrcl}
b_{1,0} & = & 100101 & \quad & c_{1,0} & = & 000100 \\
b_{1,1} & = & 100010 & & c_{1,1} & = & 000100 \\
b_{2,0} & = & 010011 & & c_{2,0} & = & 000010 \\
b_{2,1} & = & 010100 & & c_{2,1} & = & 000010 \\
b_{3,0} & = & 001101 & & c_{3,0} & = & 000001 \\
b_{3,1} & = & 001010 & & c_{3,1} & = & 000001 \\
S & = & 111333 & & & &
\end{array}$$

Satisfying assignment: $x_1 = F$, $x_2 = F$, and $x_3 = T$

$$\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

$$\begin{array}{rcl@{\qquad\qquad}rcl}
b_{1,0} & = & 100101 & c_{1,0} & = & 000100 \\
b_{1,1} & = & 100010 & c_{1,1} & = & 000100 \\
b_{2,0} & = & 010011 & c_{2,0} & = & 000010 \\
b_{2,1} & = & 010100 & c_{2,1} & = & 000010 \\
b_{3,0} & = & 001101 & c_{3,0} & = & 000001 \\
b_{3,1} & = & 001010 & c_{3,1} & = & 000001 \\
S & = & 111333 & & &
\end{array}$$

Satisfying assignment: $x_1 = F$, $x_2 = F$, and $x_3 = T$

Choose:

$$\begin{array}{rcll}
b_{1,0} & = & 100101 & (x_1 = F) \\
b_{2,0} & = & 010011 & (x_2 = F) \\
b_{3,1} & = & \underline{001010} & (x_3 = T) \\
 & & 111122 &
\end{array}$$

$$\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

$$\begin{array}{rcl@{\qquad\qquad}rcl}
b_{1,0} & = & 100101 & c_{1,0} & = & 000100 \\
b_{1,1} & = & 100010 & c_{1,1} & = & 000100 \\
b_{2,0} & = & 010011 & c_{2,0} & = & 000010 \\
b_{2,1} & = & 010100 & c_{2,1} & = & 000010 \\
b_{3,0} & = & 001101 & c_{3,0} & = & 000001 \\
b_{3,1} & = & 001010 & c_{3,1} & = & 000001 \\
S & = & 111333 & & &
\end{array}$$

Satisfying assignment: $x_1 = F$, $x_2 = F$, and $x_3 = T$

Choose:

$$\begin{array}{rcll}
b_{1,0} & = & 100101 & (x_1 = F) \\
b_{2,0} & = & 010011 & (x_2 = F) \\
b_{3,1} & = & \underline{001010} & (x_3 = T) \\
 & & 111122 &
\end{array}$$

The last three digits of this sum, 122, correspond to the number of literals satisfied in each clause: $\neg x_1$ is satisfied C_1, $\neg x_2$ and x_3 are satisfied C_2, and $\neg x_1$ and $\neg x_2$ are satisfied C_3.

$$\phi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

$$\begin{array}{rcl} b_{1,0} & = & 100101 \\ b_{1,1} & = & 100010 \\ b_{2,0} & = & 010011 \\ b_{2,1} & = & 010100 \\ b_{3,0} & = & 001101 \\ b_{3,1} & = & 001010 \\ S & = & 111333 \end{array} \qquad \begin{array}{rcl} c_{1,0} & = & 000100 \\ c_{1,1} & = & 000100 \\ c_{2,0} & = & 000010 \\ c_{2,1} & = & 000010 \\ c_{3,0} & = & 000001 \\ c_{3,1} & = & 000001 \end{array}$$

Satisfying assignment: $x_1 = F$, $x_2 = F$, and $x_3 = T$

Choose:

$$\begin{array}{rcll} b_{1,0} & = & 100101 & (x_1 = F) \\ b_{2,0} & = & 010011 & (x_2 = F) \\ b_{3,1} & = & \underline{001010} & (x_3 = T) \\ & & 111122 & \end{array}$$

The last three digits of this sum, 122, correspond to the number of literals satisfied in each clause: $\neg x_1$ is satisfied C_1, $\neg x_2$ and x_3 are satisfied C_2, and $\neg x_1$ and $\neg x_2$ are satisfied C_3.

To finish, we also choose $c_{1,0}$, $c_{1,1}$, $c_{2,0}$, and $c_{3,0}$ to reach S.

We must show that $\phi \in \mathrm{3SAT} \Rightarrow (L, S) \in \mathrm{SUBSET\text{-}SUM}$. So assume that ϕ is satisfied by an assignment τ to the variables $x_1, \ldots, x_n$. We define a sublist L' of L as follows.

We must show that $\phi \in \mathrm{3SAT} \Rightarrow (L, S) \in \mathrm{SUBSET\text{-}SUM}$. So assume that ϕ is satisfied by an assignment τ to the variables $x_1, \ldots, x_n$. We define a sublist L' of L as follows.

For each j, $1 \leq j \leq n$,

- If $\tau(x_j) = F$, then put $b_{j,0}$ in L'.
- If $\tau(x_j) = T$, then put $b_{j,1}$ in L'.

We must show that $\phi \in \mathrm{3SAT} \Rightarrow (L, S) \in \mathrm{SUBSET\text{-}SUM}$. So assume that ϕ is satisfied by an assignment τ to the variables $x_1, \ldots, x_n$. We define a sublist L' of L as follows.

For each j, $1 \leq j \leq n$,

- If $\tau(x_j) = F$, then put $b_{j,0}$ in L'.
- If $\tau(x_j) = T$, then put $b_{j,1}$ in L'.

For each i, $1 \leq i \leq m$,

- If τ satisfies 3 literals in C_i, do nothing.
- If τ satisfies 2 literals in C_i, put $c_{i,0}$ in L'.
- If τ satisfies 1 literal in C_i, put both $c_{i,0}$ and $c_{i,1}$ in L'.

We must show that $\phi \in \mathrm{3SAT} \Rightarrow (L, S) \in \mathrm{SUBSET\text{-}SUM}$. So assume that ϕ is satisfied by an assignment τ to the variables $x_1, \ldots, x_n$. We define a sublist L' of L as follows.

For each j, $1 \leq j \leq n$,

- If $\tau(x_j) = F$, then put $b_{j,0}$ in L'.
- If $\tau(x_j) = T$, then put $b_{j,1}$ in L'.

For each i, $1 \leq i \leq m$,

- If τ satisfies 3 literals in C_i, do nothing.
- If τ satisfies 2 literals in C_i, put $c_{i,0}$ in L'.
- If τ satisfies 1 literal in C_i, put both $c_{i,0}$ and $c_{i,1}$ in L'.

Then $\sum_{a \in L'} a = S$. Therefore $(L, S) \in \mathrm{SUBSET\text{-}SUM}$.

Now we must show that $(L, S) \in \mathrm{SUBSET\text{-}SUM} \Rightarrow \phi \in \mathrm{3SAT}$.
Suppose that L' is a sublist of L that sums to S.

Now we must show that $(L, S) \in \mathrm{SUBSET\text{-}SUM} \Rightarrow \phi \in \mathrm{3SAT}$. Suppose that L' is a sublist of L that sums to S.

We make two observations.

- We know that for each j, $1 \leq j \leq n$, L' has exactly one of $b_{j,0}$ or $b_{j,1}$. There's no other way to get a one in each column. Picking neither gives us a 0, and picking both gives us a 2.

Now we must show that $(L, S) \in \mathrm{SUBSET\text{-}SUM} \Rightarrow \phi \in \mathrm{3SAT}$. Suppose that L' is a sublist of L that sums to S.

We make two observations.

- We know that for each j, $1 \leq j \leq n$, L' has exactly one of $b_{j,0}$ or $b_{j,1}$. There's no other way to get a one in each column. Picking neither gives us a 0, and picking both gives us a 2.
- Furthermore, we don't have to worry about carry-over from another column because we are working in decimal notation and the largest possible sum in a column is 5.

Now we must show that $(L, S) \in \text{SUBSET-SUM} \Rightarrow \phi \in \text{3SAT}$. Suppose that L' is a sublist of L that sums to S.

We make two observations.

- We know that for each j, $1 \leq j \leq n$, L' has exactly one of $b_{j,0}$ or $b_{j,1}$. There's no other way to get a one in each column. Picking neither gives us a 0, and picking both gives us a 2.
- Furthermore, we don't have to worry about carry-over from another column because we are working in decimal notation and the largest possible sum in a column is 5.

We define our assignment τ as follows.

$$\tau(x_j) = \begin{cases} T & \text{if } L' \text{ contains } b_{j,1} \\ F & \text{if } L' \text{ contains } b_{j,0} \end{cases}$$

We now verify that τ satisfies ϕ.

For each i, $1 \leq i \leq m$, the $(n+i)^{\mathrm{th}}$ digit in the sum of L' is 3, so in L' there must be some $b_{j,0}$ or $b_{j,1}$ that has the $(n+i)^{\mathrm{th}}$ digit equal to 1. This is because from $c_{i,0}$ and $c_{i,1}$ we can get at most a sum of 2 in the $(n+i)^{\mathrm{th}}$ column, so we must get at least one 1 from the $b_{j,k}$'s.

We now verify that τ satisfies ϕ.

For each i, $1 \le i \le m$, the $(n+i)^{\mathrm{th}}$ digit in the sum of L' is 3, so in L' there must be some $b_{j,0}$ or $b_{j,1}$ that has the $(n+i)^{\mathrm{th}}$ digit equal to 1. This is because from $c_{i,0}$ and $c_{i,1}$ we can get at most a sum of 2 in the $(n+i)^{\mathrm{th}}$ column, so we must get at least one 1 from the $b_{j,k}$'s.

- Suppose it is a $b_{j,0}$. Then $\neg x_j$ is a literal in clause C_i and $\tau(x_j) = F$, so our assignment satisfies that clause.

We now verify that τ satisfies ϕ.

For each i, $1 \leq i \leq m$, the $(n+i)^{\mathrm{th}}$ digit in the sum of L' is 3, so in L' there must be some $b_{j,0}$ or $b_{j,1}$ that has the $(n+i)^{\mathrm{th}}$ digit equal to 1. This is because from $c_{i,0}$ and $c_{i,1}$ we can get at most a sum of 2 in the $(n+i)^{\mathrm{th}}$ column, so we must get at least one 1 from the $b_{j,k}$'s.

- Suppose it is a $b_{j,0}$. Then $\neg x_j$ is a literal in clause C_i and $\tau(x_j) = F$, so our assignment satisfies that clause.
- Similarly, suppose it is a $b_{j,1}$. Then x_j is a literal in clause C_i and $\tau(x_j) = T$, so our assignment satisfies that clause.

We now verify that τ satisfies ϕ.

For each i, $1 \leq i \leq m$, the $(n+i)^{\mathrm{th}}$ digit in the sum of L' is 3, so in L' there must be some $b_{j,0}$ or $b_{j,1}$ that has the $(n+i)^{\mathrm{th}}$ digit equal to 1. This is because from $c_{i,0}$ and $c_{i,1}$ we can get at most a sum of 2 in the $(n+i)^{\mathrm{th}}$ column, so we must get at least one 1 from the $b_{j,k}$'s.

- Suppose it is a $b_{j,0}$. Then $\neg x_j$ is a literal in clause C_i and $\tau(x_j) = F$, so our assignment satisfies that clause.
- Similarly, suppose it is a $b_{j,1}$. Then x_j is a literal in clause C_i and $\tau(x_j) = T$, so our assignment satisfies that clause.

Therefore, τ satisfies all the clauses of ϕ, so ϕ is satisfiable, and $\phi \in \mathrm{3SAT}$.

We now verify that τ satisfies ϕ.

For each i, $1 \leq i \leq m$, the $(n+i)^{\text{th}}$ digit in the sum of L' is 3, so in L' there must be some $b_{j,0}$ or $b_{j,1}$ that has the $(n+i)^{\text{th}}$ digit equal to 1. This is because from $c_{i,0}$ and $c_{i,1}$ we can get at most a sum of 2 in the $(n+i)^{\text{th}}$ column, so we must get at least one 1 from the $b_{j,k}$'s.

- Suppose it is a $b_{j,0}$. Then $\neg x_j$ is a literal in clause C_i and $\tau(x_j) = F$, so our assignment satisfies that clause.
- Similarly, suppose it is a $b_{j,1}$. Then x_j is a literal in clause C_i and $\tau(x_j) = T$, so our assignment satisfies that clause.

Therefore, τ satisfies all the clauses of ϕ, so ϕ is satisfiable, and $\phi \in \mathrm{3SAT}$.

Since the construction is polynomial-time computable, we have shown that $\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{SUBSET\text{-}SUM}$. $\square$

NP-Completeness Summary

Let us review the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.

NP-Completeness Summary

Let us review the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.
2. Choose some NP-complete problem A that is a candidate for reduction to B. Often we try to find a problem that is similar to B or use $3\mathrm{SAT}$.

NP-Completeness Summary

Let us review the method of showing a problem B is NP-complete.

1. Show that $B \in \mathrm{NP}$. It is usually easy to show this, but it is a step that shouldn't be overlooked.
2. Choose some NP-complete problem A that is a candidate for reduction to B. Often we try to find a problem that is similar to B or use $3\mathrm{SAT}$.
3. Reduce A to B.

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.

$$I \mapsto I'$$

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
   $$I \mapsto I'$$
   Then we need to prove that for every instance I of A,
   $$I \in A \iff I' \in B.$$

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
   $$I \mapsto I'$$
   Then we need to prove that for every instance I of A,
   $$I \in A \iff I' \in B.$$
   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.
   $$I \mapsto I'$$
   Then we need to prove that for every instance I of A,
   $$I \in A \iff I' \in B.$$
   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.
   - To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.

3. Reduce A to B.
   For this, we need to define a mapping of instances I of A to instances I' of B.

$$I \mapsto I'$$

   Then we need to prove that for every instance I of A,

$$I \in A \iff I' \in B.$$

   This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.
   - To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.
   - To show the converse, assume that $I' \in B$. Then there is some witness w' for I'. We use w' to construct a witness w that shows $I \in A$.

3. Reduce A to B.
For this, we need to define a mapping of instances I of A to instances I' of B.

$$I \mapsto I'$$

Then we need to prove that for every instance I of A,

$$I \in A \iff I' \in B.$$

This involves showing two implications: $I \in A \implies I' \in B$ and $I' \in B \implies I \in A$.

- To show that $I \in A \implies I' \in B$, assume that $I \in A$. Then there is some witness w for I that shows I is a member of A. We use this witness to construct a witness w' that shows $I' \in B$.
- To show the converse, assume that $I' \in B$. Then there is some witness w' for I'. We use w' to construct a witness w that shows $I \in A$.

Finally, we have to verify that the mapping is polynomial-time computable. Usually this doesn't require formal analysis of the algorithm.

Summary of NP-completeness reductions: