



NP-Completeness Reductions

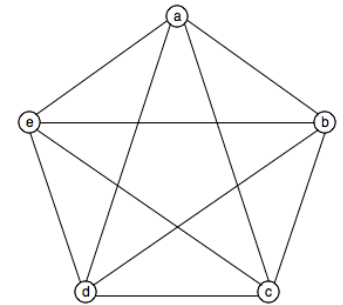
CS240

Spring 2022

Rui Fan

CLIQUE is NP-complete

- CLIQUE Given a graph with n nodes, is there a clique with $n/3$ nodes?
 - I.e., are there $n/3$ nodes s.t. they're all connected to each other?
 - Actually, the CLIQUE problem asks if there's a k -clique, for an arbitrary k . But we consider $k=n/3$ for simplicity.
- $\text{CLIQUE} \in \text{NP}$.
 - The witness is a purported $n/3$ -clique.
 - The verifier just checks there are $n/3$ nodes, and they're all connected.





CLIQUE is NP-complete

- Show some NP-complete problem reduces to CLIQUE.
 - The problem you reduce from has to be NP-complete, not just in NP.
 - Note, you're reducing from the NPC problem to your problem, not the other way around.
 - You can choose any NP-complete problem to reduce from.
 - Decide on the right problem can make the task a lot easier. But this takes careful thought.

CLIQUE is NP-complete

■ 3-CNF-SAT

- Given a Boolean formula that's an AND of ORs, where each OR has **3 literals**, is it satisfiable?
- $(A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg D) \in 3\text{-CNF-SAT}$.
 - Set $A=B=C=\text{true}$, $D=\text{false}$.
- Each OR unit is called a **clause**. Each literal is either a variable or its negation.
- A special kind of SAT. SAT allows other formula types, besides ANDs of ORs, and allows any number of variables per clause.

■ Assume we've already proven 3-CNF-SAT is NP-complete.

■ We show $3\text{-CNF-SAT} \leq_p \text{CLIQUE}$.

■ The reduction says that given a 3-CNF-SAT formula ϕ , we can create in polytime a graph G , such that ϕ is satisfiable if and only if G has an $n/3$ -clique.

- This is actually quite remarkable. Why should a graph be related to a formula?
- But we'll see how to construct a special graph that captures the satisfiability of a 3-CNF formula.



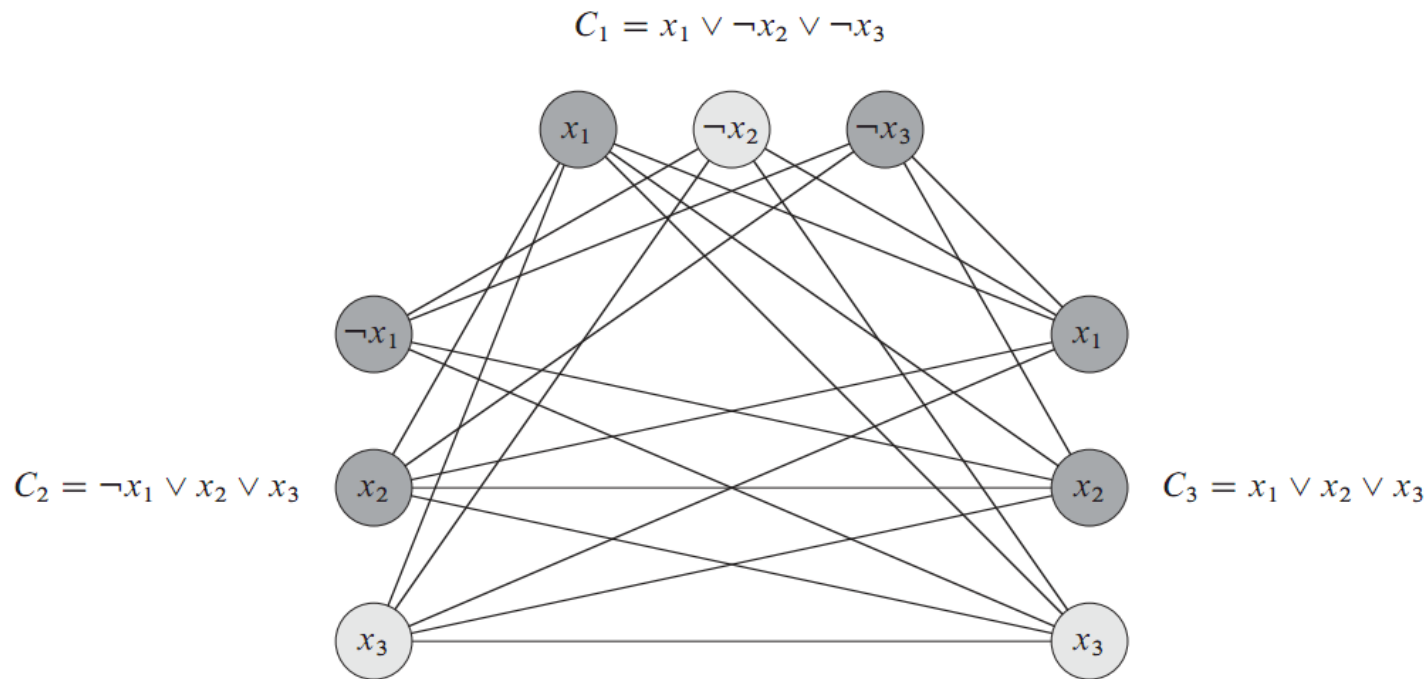
Reducing 3-CNF-SAT to CLIQUE

- Let ϕ be a 3-CNF formula with m clauses.
- Let C be a clause in ϕ . Then C has 3 literals.
 - Make 3 vertices in G corresponding to the literals.
 - So G has $3m$ vertices total.
 - Let n be the number of nodes in G . Then $m = n/3$.
- Now, add in an edge between two vertices u, v if both conditions below hold.
 - u, v correspond to literals from different clauses of ϕ .
 - The literals corresponding to u and v are not negations of each other.
 - We say u and v are **consistent**.

Reducing 3-CNF-SAT to CLIQUE

- 3 vertices for each clause.
- For vertices u, v , add edge (u, v) if u, v are from different clauses, and are consistent (not negations of each other).

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



Source: Introduction to Algorithms, Cormen et al



Proving the reduction works

- We first need to show the reduction runs in polytime.
 - Yes. If there are n clauses, the reduction takes $O(n^2)$ time.
- Recall the graph has $n = 3m$ nodes, so $m = n/3$.
- Show $\phi \in 3\text{-CNF-SAT} \Leftrightarrow G \in m\text{-CLIQUE}$.
 - (\Rightarrow) If ϕ has a satisfying assignment, then G has an m clique.
 - (\Leftarrow) If G has an m clique, then ϕ is satisfiable.

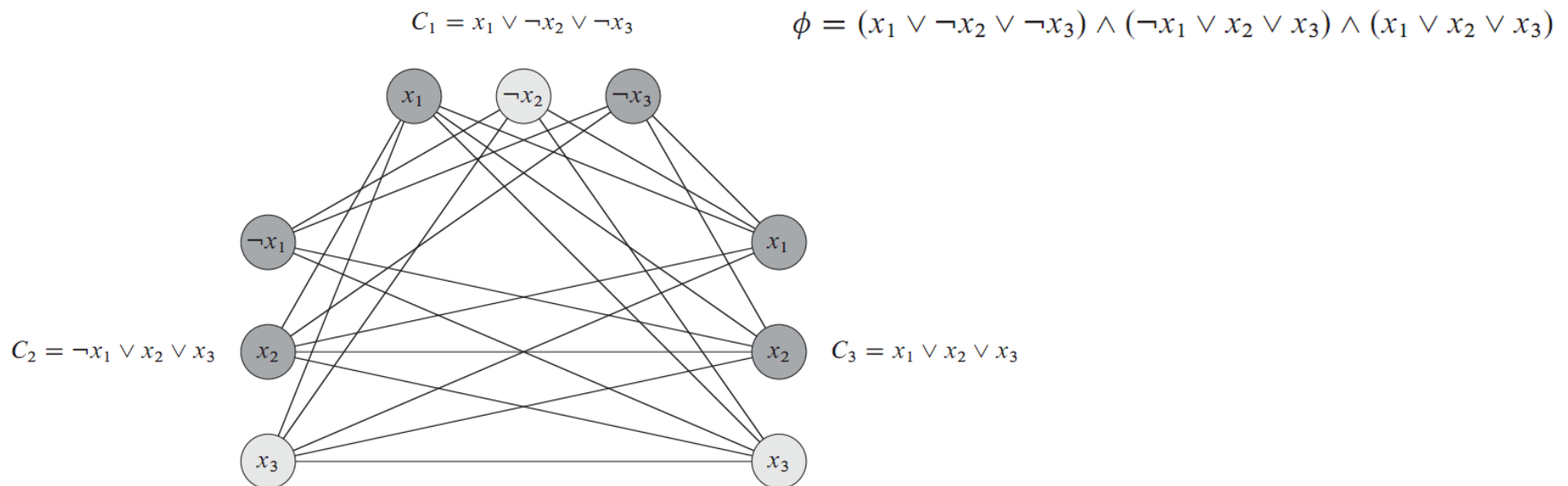


\exists sat. assignment $\Rightarrow \exists$ m clique

- In the satisfying assignment, every clause has to be true, since we AND them.
- In each clause, at least one literal has to be true, since we OR them.
- So for each clause, pick a true literal.
 - We pick $m = n/3$ literals.
- The true literal corresponds to a vertex in the graph.
 - Pick m vertices corresponding to the m literals we picked.
- **Claim** The selected vertices form an m -clique.
- **Proof** Consider any 2 vertices u, v we selected.
 - u, v come from different triples.
 - Because they come from literals from different clauses.

\exists sat. assignment $\Rightarrow \exists$ m clique

- **Proof ctd** u, v are consistent. I.e. they don't correspond to a literal in one clause, and its negation in another clause.
 - Because we only picked true literals.
 - So there's an edge (u, v) , by construction.
 - So any 2 of the m selected vertices are connected. So the vertices are an m -clique.
- **Ex** ϕ has a satisfying assignment $x_1 = x_2 = x_3 = T$.
 - The corresponding nodes form a 3-clique.





\exists m clique $\Rightarrow \exists$ sat. assignment

- Consider the m vertices in the clique.
- None of the vertices come from literals in the same clause in ϕ .
 - For any pair of vertices, they're connected.
 - There are no edges between vertices from the same clause.
- None of the vertices correspond to a literal and its negation in ϕ .
 - We don't add edges between such vertices.
- For the literals corresponding to the clique vertices, set all of them to be true in the formula.
 - This is a valid assignment, since we never set a literal and its negation both to true.
 - We have one true literal per clause.
 - So every clause is true.
 - So the formula is true.



CLIQUE is NP-complete

- We've shown $\text{CLIQUE} \in \text{NP}$.
- We've shown $3\text{-CNF-SAT} \leq_p \text{CLIQUE}$.
 - We found a polytime reduction, constructing a graph G s.t. for every 3-CNF-SAT formula ϕ
 - If ϕ is satisfiable, G has an $n/3$ -clique.
 - If G has an $n/3$ -clique, then ϕ is satisfiable.
- So CLIQUE is NP-complete.



SUBSET-SUM is NP-complete

- Recall that in SUBSET-SUM, we are given a set of numbers $S = \{s_1, \dots, s_n\}$ and a target value t , and we want to find a subset $S' \subseteq S$ summing to t , i.e. $\sum_{s \in S'} s = t$.
- SUBSET-SUM \in NP.
 - The witness is a subset S' of S .
 - The verifier simply checks that S' sums up to t .
- To show SUBSET-SUM is NP-complete, we show 3-CNF-SAT \leq_P SUBSET-SUM.
 - 3-CNF-SAT is a flexible problem used in many reductions.
- Given a 3-CNF formula ϕ , we construct in polytime a set S and target t s.t.
 - ϕ is satisfiable implies there's a subset of S summing to t .
 - If there's a subset of S summing to t , then ϕ is satisfiable.
 - This construction is the polytime reduction \leq_P .

The reduction

- Suppose ϕ contains n variables x_1, \dots, x_n and k clauses C_1, \dots, C_k .
 - Assume WLOG that no clause contains a variable and its negation, since those clauses are automatically satisfied.
- The reduction creates a set S with $2n+2k$ numbers, two for each variable and clause.
 - Each number has $n+k$ digits, with one digit corresponding to each variable and each clause.
 - The numbers are in base 10.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .

The reduction

- For each variable x_i , S contains two numbers v_i and v'_i .
 - v_i and v'_i both have a 1 in x_i 's digit, and 0's in all the other variable digits.
 - If x_i appears in clause C_j , then the j 'th clause digit in v_i is 1.
 - If $\neg x_i$ appears in clause C_j , then the j 'th clause digit in v'_i is 1.
 - All other clause digits in v_i and v'_i are 0.
- For each clause C_j , S contains two numbers s_j and s'_j .
 - s_j has a 1 in the C_j digit, and s'_j has a 2 in this digit.
 - s_j and s'_j are 0's elsewhere.
- Target t is 1 in all the variable digits and 4 in all the clause digits.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S. Last row is target t .

$$\phi \in 3\text{-CNF-SAT} \Rightarrow (S, t) \in \text{SUBSET-SUM}$$

- Suppose there's a satisfying assignment ρ to ϕ .
- We form a subset S' of S summing to t based on ρ .
 - If $x_i = T$ in ρ , include v_i in S' .
 - If $x_i = F$ in ρ , include v'_i in S' .
- **Claim 1** Any variable digit x_i sums to 1.
 - Either v_i or v'_i is in S' , but not both.
 - Both v_i and v'_i cause digit x_i to be 1.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .
- Lightly shaded rows sum to t , and correspond to a satisfying assignment $x_1 = F, x_2 = F, x_3 = T$.

$$\phi \in 3\text{-CNF-SAT} \Rightarrow (S, t) \in \text{SUBSET-SUM}$$

■ **Claim 2** Any clause digit C_j sums to ≥ 1 .

- Since ρ is a satisfying assignment, C_j must have one true literal in ρ .
- If the literal is x_i , then $x_i = T$ in ρ , and $v_i \in S'$.
 - v_i has a 1 in clause digit C_j , by construction.
- If the literal is $\neg x_i$, then $x_i = F$ in ρ , and $v'_i \in S'$.
 - v'_i has a 1 in clause digit C_j , by construction.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .
- Lightly shaded rows sum to t , and correspond to a satisfying assignment $x_1 = F, x_2 = F, x_3 = T$.

$$\phi \in 3\text{-CNF-SAT} \Rightarrow (S, t) \in \text{SUBSET-SUM}$$

- **Claim 3** Any clause digit C_j sums to ≤ 3 .
 - C_j includes 3 literals.
 - The v or v' corresponding to each literal is either in S' or not.
 - If it's in S' , it contributes 1 to C_j .
- Each clause digit C_j sums to between 1 to 3 using the current elements of S' .
 - Add s_j to S' if the sum is 3.
 - Add s'_j to S' if the sum is 2.
 - Add $\{s_j, s'_j\}$ to S' if the sum is 1.
- Now digit C_j sums to 4.
- Since all the variable digits sum to 1 by Claim 1, we have that S' sums to t .
- Thus, ϕ is satisfiable \Rightarrow there's a subset of S summing to t .

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .
- Lightly shaded rows sum to t , and correspond to a satisfying assignment $x_1 = F, x_2 = F, x_3 = T$.

$$(S, t) \in \text{SUBSET-SUM} \Rightarrow \phi \in \text{3-CNF-SAT}$$

- Assume there's a subset S' summing to t .
 - We use S' to form a satisfying assignment ρ for ϕ .
- Notice the largest sum in any digit is 6.
 - Each variable digit sums to ≤ 2 .
 - Each clause digit has three 1's among the v_i, v'_i values, since the clause contains 3 literals.
 - The s_j, s'_j values also sum to ≤ 3 .
- Thus, there are no “carries” when we add values from S , i.e. the sum in each column comes only from values in that column.
- So since S' sums to 1 in the variable digits, it contains either v_i or v'_i , but not both.
- If $v_i \in S'$, set $x_i = T$. If $v'_i \in S'$, set $x_i = F$.
 - Call this assignment ρ . Note ρ is valid, i.e. either $x_i = T$ or $x_i = F$, but not both.
 - We show ρ satisfies ϕ .

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .
- Lightly shaded rows sum to t , and correspond to a satisfying assignment $x_1 = F, x_2 = F, x_3 = T$.

$$(S, t) \in \text{SUBSET-SUM} \Rightarrow \phi \in 3\text{-CNF-SAT}$$

- Consider a clause C_j .
 - Since C_j 's clause digit in t is 4, and s_j and s_j' sum to ≤ 3 in this digit, S' must contain either a v_i or v_i' that has a 1 in clause digit C_j .
 - If $v_i \in S'$ and v_i has 1 in digit C_j , then x_i occurs in clause C_j .
 - Since we set $x_i = T$ in ρ , clause C_j is satisfied.
 - If $v_i' \in S'$ and v_i' has 1 in digit C_j , then $\neg x_i$ occurs in clause C_j .
 - Since we set $x_i = F$ in ρ , clause C_j is satisfied.
- In this way, all clauses satisfied.
 - So $\phi \in 3\text{-CNF-SAT}$.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v_1'	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v_3'	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s_3'	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

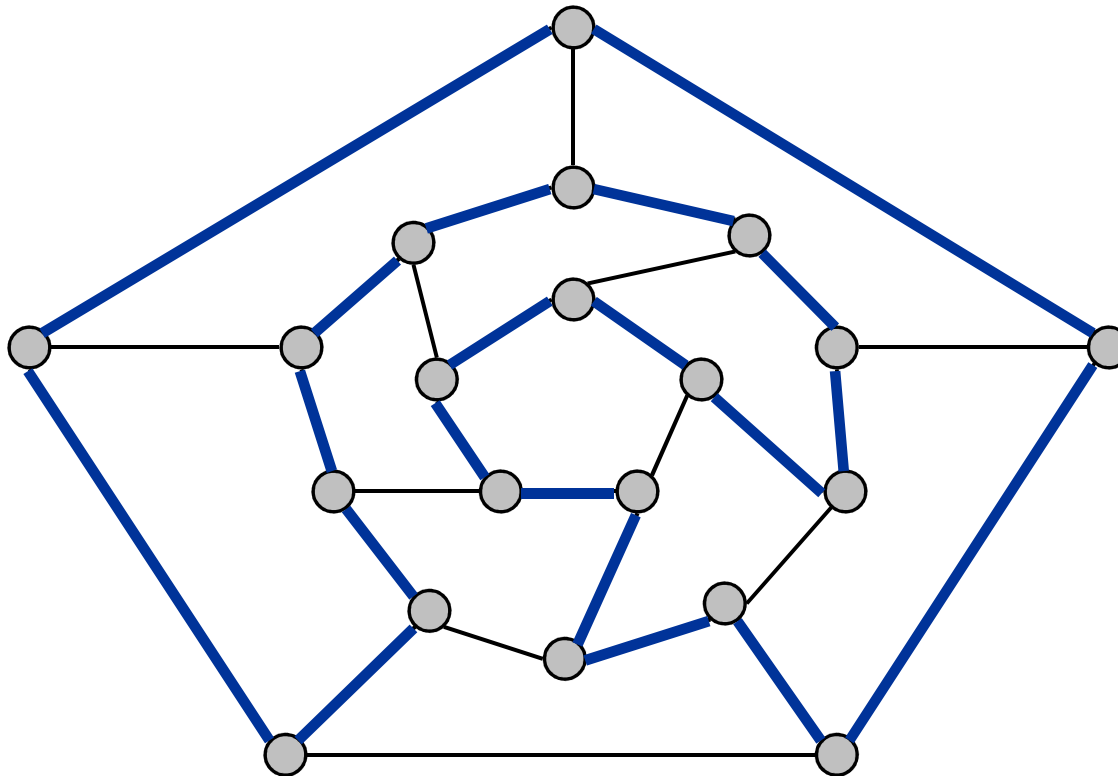
- SUBSET-SUM instance for $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- Each row except last represents a base-10 number in S . Last row is target t .
- Lightly shaded rows sum to t , and correspond to a satisfying assignment $x_1 = F, x_2 = F, x_3 = T$.

SUBSET-SUM is NP-complete

- The reduction runs in polynomial time.
 - It creates $2n+2k+1$ numbers, each with $n+k$ digits. Each digit is computed in $O(1)$ time.
- So $3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}$. Since $\text{SUBSET-SUM} \in \text{NP}$, then SUBSET-SUM is NP-complete.
- But didn't we show $\text{SUBSET-SUM} \in P$ by dynamic programming?
 - The dynamic program ran in $O(nW)$ time, where n is the number of elements in S , and W is the sum of the elements.
 - So did we prove $P=NP$!?
- No ☹, because $O(nW)$ is not polynomial in the input size.
 - The input has n numbers, each with $O(\log W)$ bits.
 - So the input size is $O(n \log W)$.
 - The running time $O(nW)$ is exponential in the input size.
 - For example, in the previous reduction, we had $2n+2k+1$ numbers, but the sum of the numbers is $W \leq (2n + 2k + 1)10^{n+k}$.

Hamiltonian Cycle

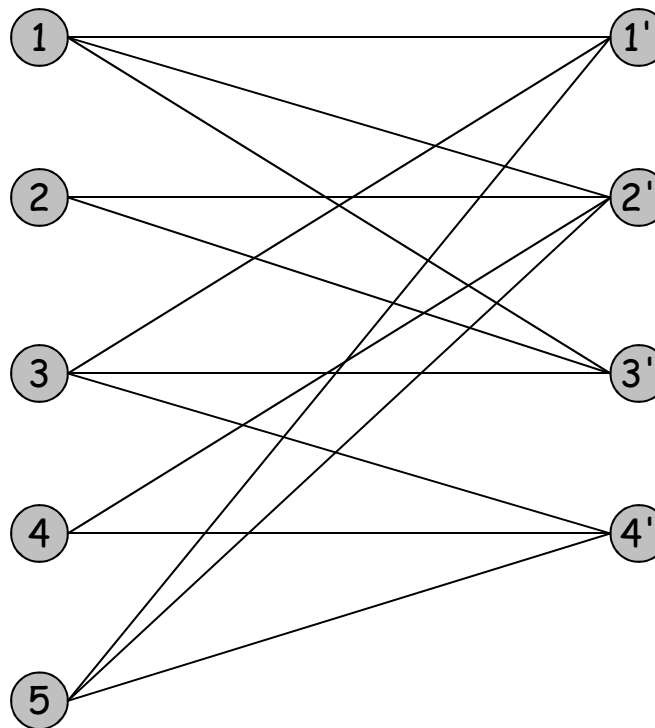
HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V .



YES: vertices and faces of a dodecahedron.

Hamiltonian Cycle

HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V .



NO: bipartite graph with odd number of nodes.

3-SAT Reduces to Directed Hamiltonian Cycle

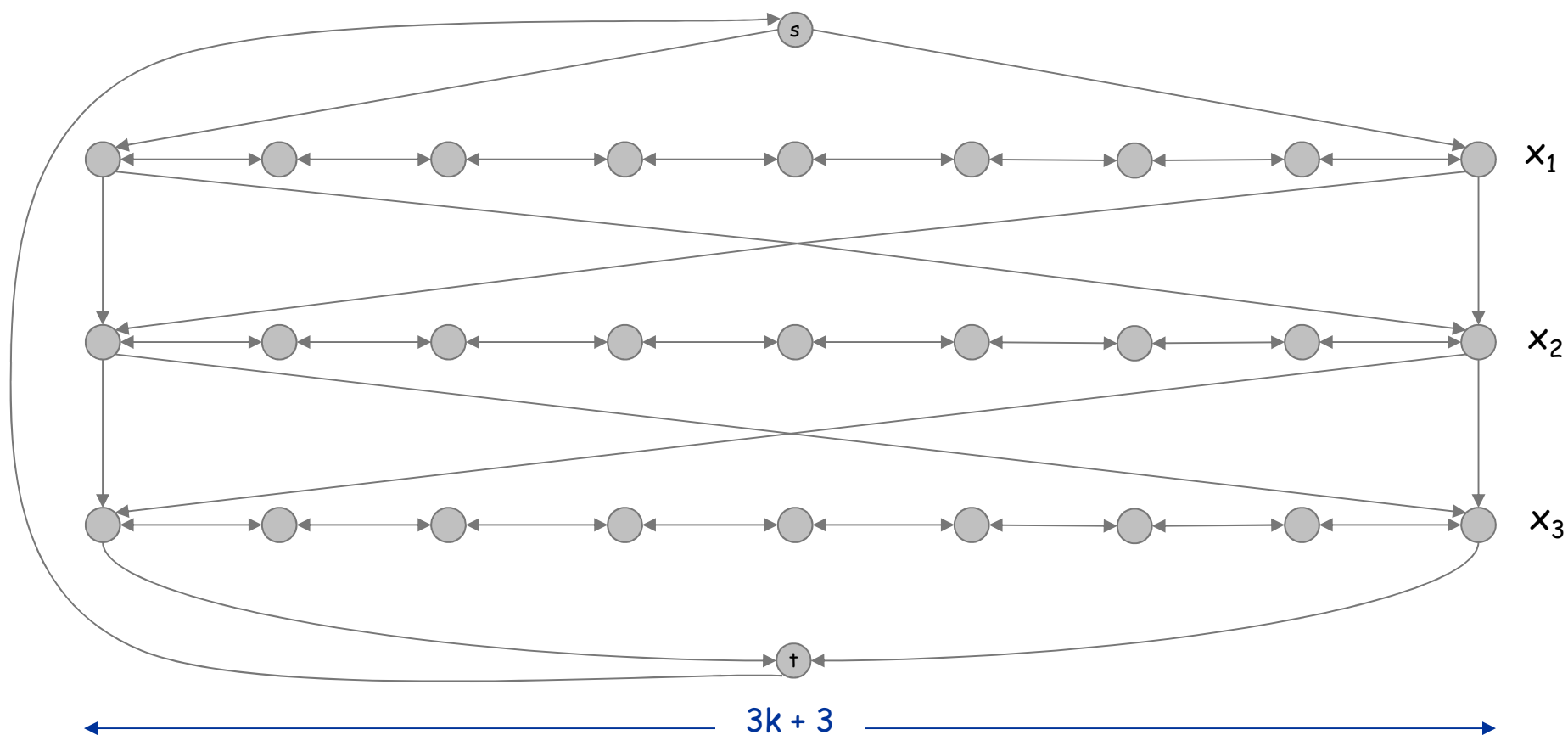
Claim. $3\text{-SAT} \leq_p \text{DIR-HAM-CYCLE}$.

Pf. Given an instance Φ of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff Φ is satisfiable.

3-SAT Reduces to Directed Hamiltonian Cycle

Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

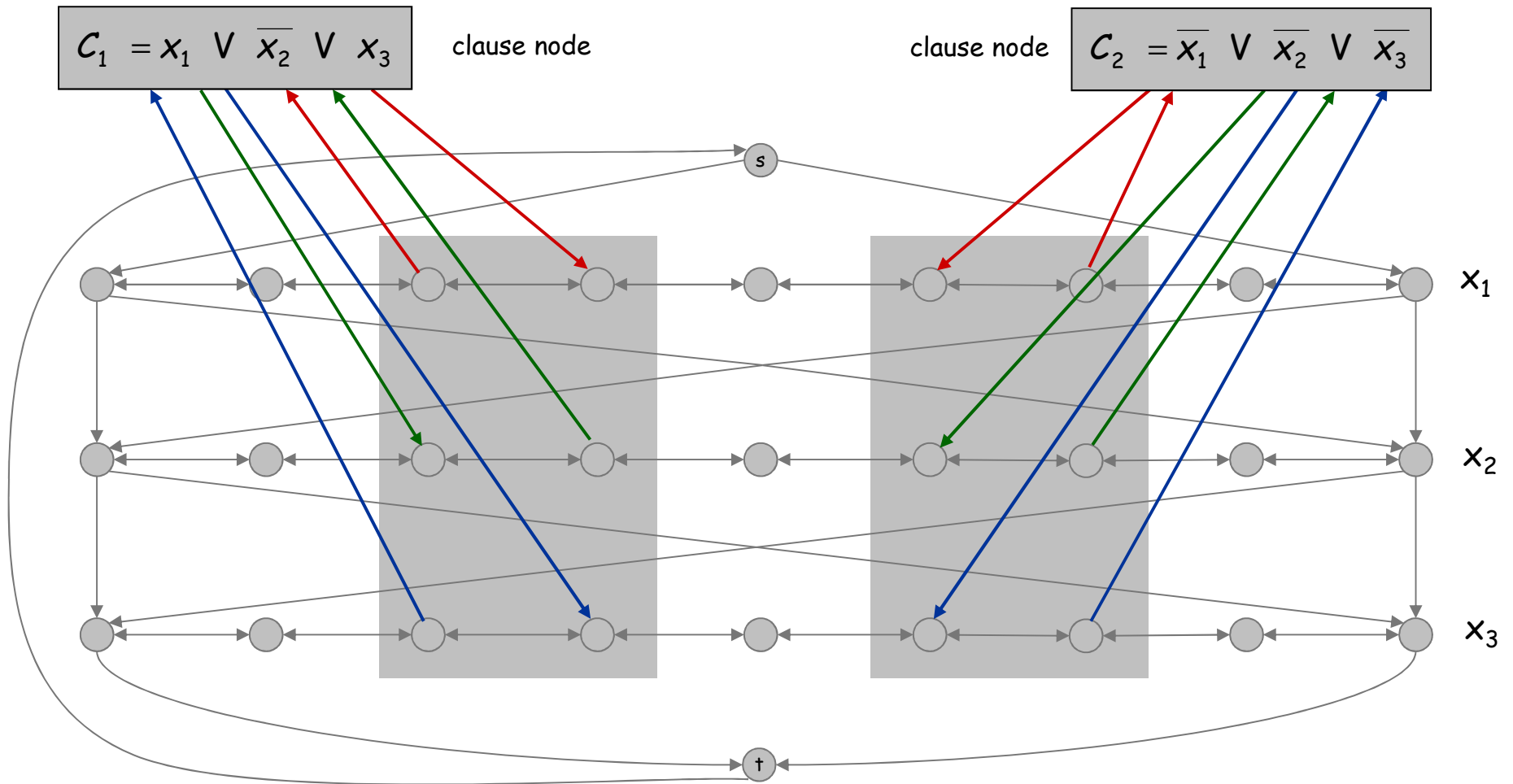
- Construct G to have 2^n Hamiltonian cycles.
- Intuition: traverse path i from left to right \Leftrightarrow set variable $x_i = 1$.



3-SAT Reduces to Directed Hamiltonian Cycle

Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

- For each clause: add a node and 6 edges.



3-SAT Reduces to Directed Hamiltonian Cycle

Claim. Φ is satisfiable iff G has a Hamiltonian cycle.

Pf. \Rightarrow

- Suppose 3-SAT instance has satisfying assignment x^* .
- Then, define Hamiltonian cycle in G as follows:
 - if $x_i^* = 1$, traverse row i from left to right
 - if $x_i^* = 0$, traverse row i from right to left
 - for each clause C_j , there will be at least one row i in which we are going in "correct" direction to splice node C_j into tour

3-SAT Reduces to Directed Hamiltonian Cycle

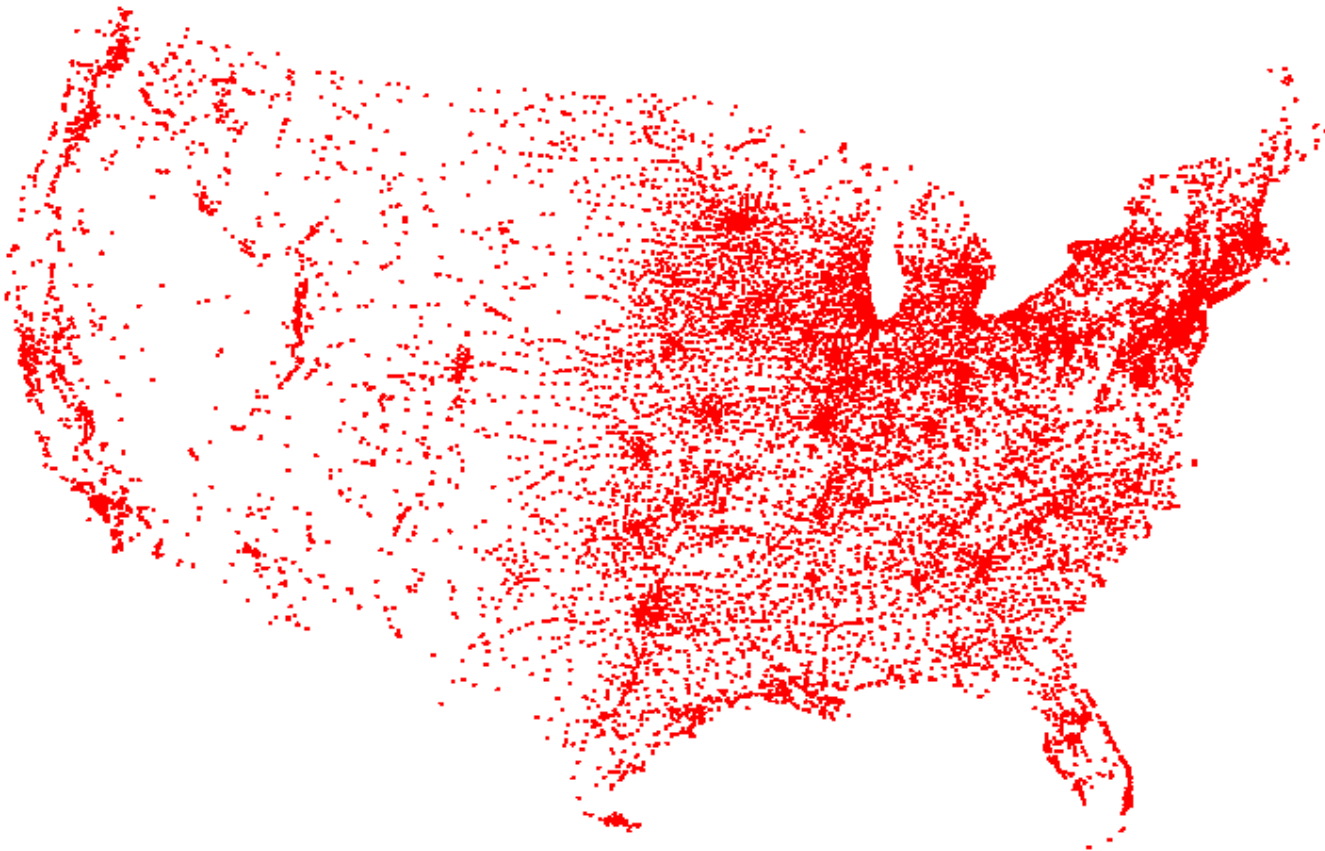
Claim. Φ is satisfiable iff G has a Hamiltonian cycle.

Pf. \Leftarrow

- Suppose G has a Hamiltonian cycle Γ .
- If Γ enters clause node C_j , it must depart on mate edge.
 - thus, nodes immediately before and after C_j are connected by an edge e in G
 - removing C_j from cycle, and replacing it with edge e yields Hamiltonian cycle on $G - \{ C_j \}$
- Continuing in this way, we are left with Hamiltonian cycle Γ' in $G - \{ C_1, C_2, \dots, C_k \}$.
- Set $x_i^* = 1$ iff Γ' traverses row i left to right.
- Since Γ visits each clause node C_j , at least one of the paths is traversed in "correct" direction, and each clause is satisfied. ▪

Traveling Salesperson Problem

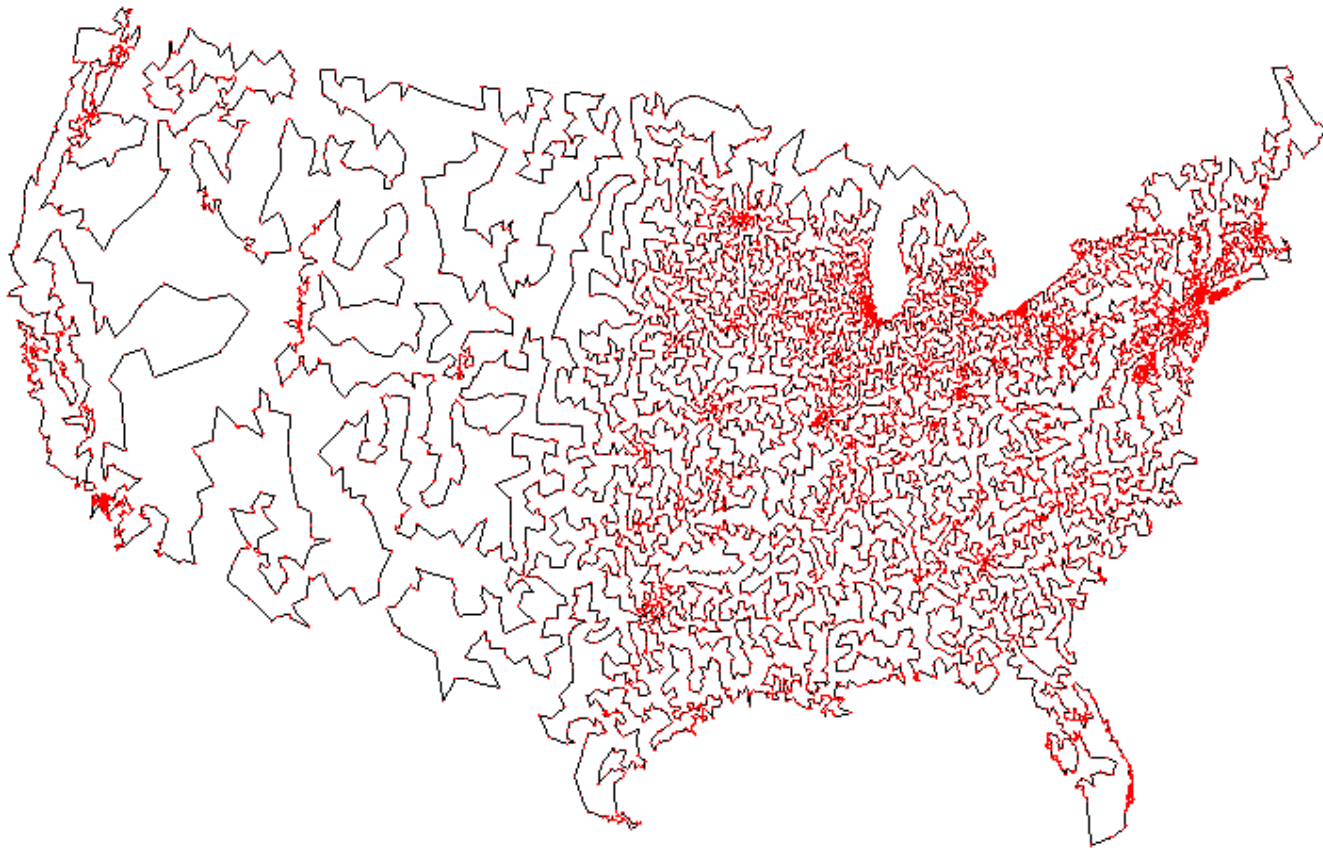
TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?



All 13,509 cities in US with a population of at least 500
Reference: <http://www.tsp.gatech.edu>

Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?



Optimal TSP tour
Reference: <http://www.tsp.gatech.edu>

Traveling Salesperson Problem

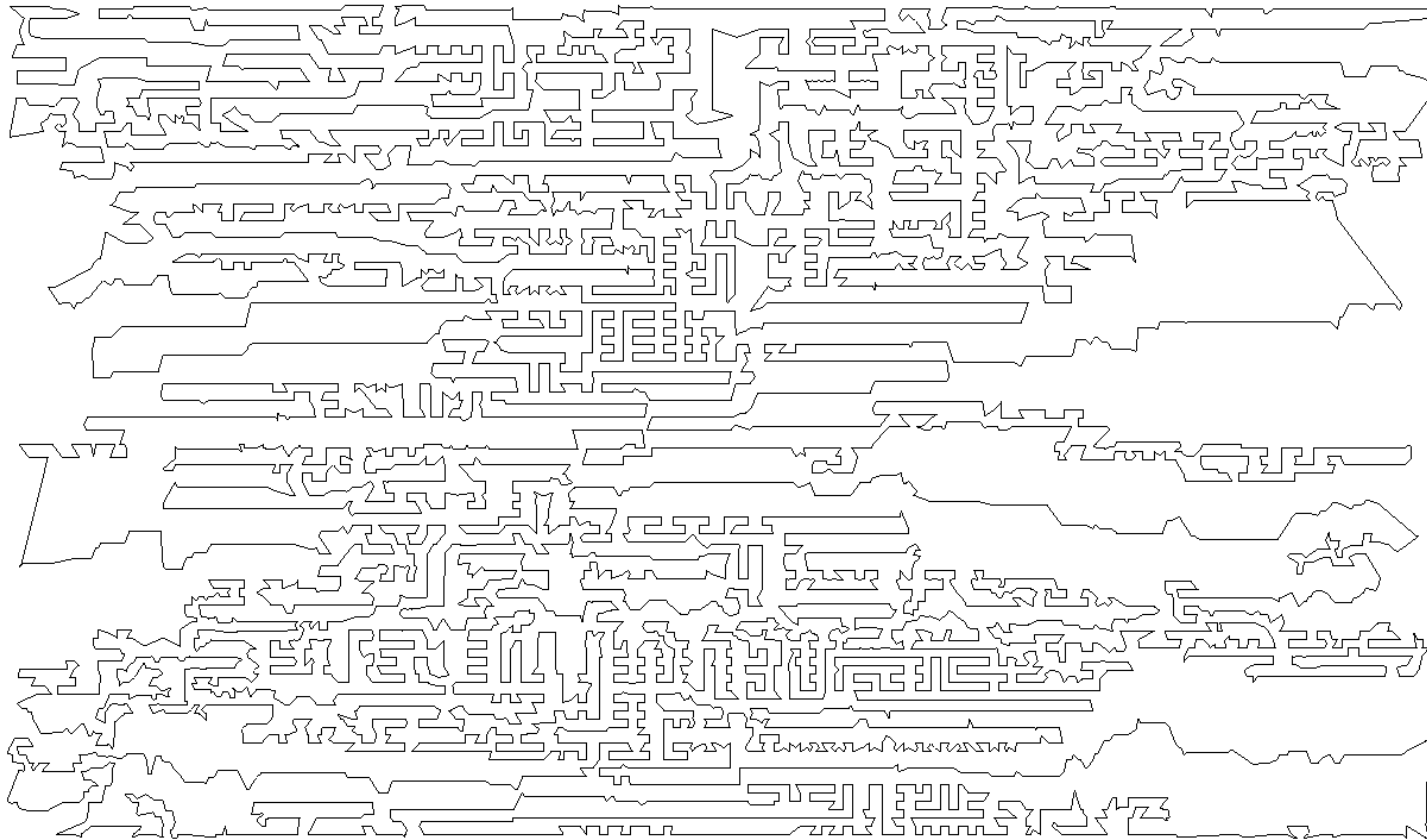
TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?



11,849 holes to drill in a programmed logic array
Reference: <http://www.tsp.gatech.edu>

Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?



Optimal TSP tour
Reference: <http://www.tsp.gatech.edu>

Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

HAM-CYCLE: given a graph $G = (V, E)$, does there exist a simple cycle that contains every node in V ?

Claim. $\text{HAM-CYCLE} \leq_p \text{TSP}$.

Pf.

- Given instance $G = (V, E)$ of HAM-CYCLE, create n cities with distance function

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length $\leq n$ iff G is Hamiltonian. ▪