Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems

NP-Complete Problems

T. M. Murali

April 28, 30, 2015



▶ Claim: If Y is NP-Complete and $X \in NP$ such that $Y \leq_P X$, then X is NP-Complete.



- ▶ Claim: If Y is NP-Complete and $X \in NP$ such that $Y \leq_P X$, then X is NP-Complete.
- Given a new problem X, a general strategy for proving it \mathcal{NP} -Complete is

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
	Proving	Other Problem	ns $\mathcal{NP} extsf{-Comple}$	te
		CO-NP NP	NP-hard	

- ▶ Claim: If Y is NP-Complete and $X \in NP$ such that $Y \leq_P X$, then X is NP-Complete.
- Given a new problem X, a general strategy for proving it NP-Complete is
 - 1. Prove that $X \in \mathcal{NP}$.
 - 2. Select a problem Y known to be \mathcal{NP} -Complete .
 - 3. Prove that $Y \leq_P X$.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
	Proving (Other Problen	ns \mathcal{NP} -Complet	e
co-NP NP NP-hard				
	(P NP	c	

- ▶ Claim: If *Y* is NP-Complete and *X* ∈ NP such that *Y* ≤_{*P*} *X*, then *X* is NP-Complete.
- Given a new problem X, a general strategy for proving it \mathcal{NP} -Complete is
 - 1. Prove that $X \in \mathcal{NP}$.
 - 2. Select a problem Y known to be \mathcal{NP} -Complete .
 - 3. Prove that $Y \leq_P X$.
- ► To prove X is NP-Complete, reduce a known NP-Complete problem Y to X. Do not prove reduction in the opposite direction, i.e., X ≤_P Y.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
Proving	a Problem	$\mathcal{NP}\text{-}Complete$	with Karp	Reduction

- 1. Prove that $X \in \mathcal{NP}$.
- 2. Select a problem Y known to be \mathcal{NP} -Complete.
- 3. Consider an arbitrary input s to problem Y. Show how to construct, in polynomial time, an input t to problem X such that
 - (a) If Y(s) = yes, then X(t) = yes and
 - (b) If X(t) = yes, then Y(s) = yes (equivalently, if Y(s) = no, then X(t) = no).

3-SAT is \mathcal{NP} -Complete

► Why is 3-SAT in NP?

3-SAT is \mathcal{NP} -Complete

- ▶ Why is 3-SAT in NP?
- CIRCUIT SATISFIABILITY $\leq_P 3$ -SAT.
 - 1. Given an instance of CIRCUIT SATISFIABILITY, create an instance of SAT, in which each clause has *at most* three variables.
 - 2. Convert this instance of SAT into one of 3-SAT.



Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

▶ Skip proof that CIRCUIT SATISFIABILITY ≤_P 3-SAT

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has \neg and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has \neg and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that x_v = x_u using clauses (x_v ∨ x_u) and (x_v ∨ x_u).
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- ▶ node v has \land and edges entering from nodes u and w: ensure $x_v = x_u \land x_w$ using clauses

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- ▶ node v has \land and edges entering from nodes u and w: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- ▶ node v has \land and edges entering from nodes u and w: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.
- Constants at sources: single-variable clauses.

- ► Given an arbitrary circuit K, associate each node v with a Boolean variable x_v.
- Encode the requirements of each gate as a clause.
- ▶ node v has ¬ and edge entering from node u: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- ▶ node v has \lor and edges entering from nodes u and w: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- ▶ node v has \land and edges entering from nodes u and w: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.
- Constants at sources: single-variable clauses.
- Output: if o is the output node, use the clause (x_o) .

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - clauses are satisfiable $\rightarrow K$ is satisfiable.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- ► Converting instance of SAT to an instance of 3-SAT.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- ► Converting instance of SAT to an instance of 3-SAT.
 - Create four new variables z_1, z_2, z_3, z_4 such that any satisfying assignment will have $z_1 = z_2 = 0$ by adding clauses

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- ► Converting instance of SAT to an instance of 3-SAT.
 - Create four new variables z_1, z_2, z_3, z_4 such that any satisfying assignment will have $z_1 = z_2 = 0$ by adding clauses $(\overline{z_i} \lor z_3 \lor z_4)$, $(\overline{z_i} \lor \overline{z_3} \lor z_4)$, $(\overline{z_i} \lor z_3 \lor \overline{z_4})$, and $(\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})$, for i = 1 and i = 2.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- ► Converting instance of SAT to an instance of 3-SAT.
 - Create four new variables z₁, z₂, z₃, z₄ such that any satisfying assignment will have z₁ = z₂ = 0 by adding clauses (z_i ∨ z₃ ∨ z₄), and (z_i ∨ z₃ ∨ z₄), for i = 1 and i = 2.
 - If a clause has a single term t, replace the clause with $(t \lor z_1 \lor z_2)$.

- ▶ Prove that *K* is equivalent to the input to SAT.
 - K is satisfiable \rightarrow clauses are satisfiable.
 - ► clauses are satisfiable → K is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- ► Converting instance of SAT to an instance of 3-SAT.
 - Create four new variables z_1, z_2, z_3, z_4 such that any satisfying assignment will have $z_1 = z_2 = 0$ by adding clauses $(\overline{z_i} \lor z_3 \lor z_4)$, $(\overline{z_i} \lor \overline{z_3} \lor z_4)$, $(\overline{z_i} \lor \overline{z_3} \lor z_4)$, $(\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})$, and $(\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})$, for i = 1 and i = 2.
 - If a clause has a single term t, replace the clause with $(t \lor z_1 \lor z_2)$.
 - If a clause has a two terms t and t', replace the clause with $t \lor t' \lor z_1$.

More \mathcal{NP} -Complete problems

- CIRCUIT SATISFIABILITY is \mathcal{NP} -Complete.
- ▶ We just showed that CIRCUIT SATISFIABILITY \leq_P 3-SAT.
- We know that
- 3-SAT \leq_P Independent Set \leq_P Vertex Cover \leq_P Set Cover
 - All these problems are in \mathcal{NP} .
 - \blacktriangleright Therefore, INDEPENDENT SET, VERTEX COVER, and SET COVER are $\mathcal{NP}\text{-}\mathsf{Complete}.$

Hamiltonian Cycle

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.

Hamiltonian Cycle

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.
- ► In a directed graph G(V, E), a cycle C is a Hamiltonian cycle if C visits each vertex exactly once.



HAMILTONIAN CYCLE **INSTANCE:** A directed graph *G*. **QUESTION:** Does *G* contain a Hamiltonian cycle?

Hamiltonian Cycle

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.
- ► In a directed graph G(V, E), a cycle C is a Hamiltonian cycle if C visits each vertex exactly once.



HAMILTONIAN CYCLE **INSTANCE:** A directed graph *G*. **QUESTION:** Does *G* contain a Hamiltonian cycle?

Hamiltonian Cycle is \mathcal{NP} -Complete

• Why is the problem in \mathcal{NP} ?

Hamiltonian Cycle is \mathcal{NP} -Complete

- Why is the problem in \mathcal{NP} ?
- Claim: $3\text{-SAT} \leq_P \text{HAMILTONIAN CYCLE.}$ \checkmark Jump to TSP

Hamiltonian Cycle is $\mathcal{NP}\text{-}\text{Complete}$

- Why is the problem in \mathcal{NP} ?
- ► Claim: $3\text{-SAT} \leq_P \text{HAMILTONIAN CYCLE}$. Jump to TSP
- ► Consider an arbitrary input to 3-SAT with variables x₁, x₂,..., x_n and clauses C₁, C₂,... C_k.
- Strategy:
 - 1. Construct a graph G with O(nk) nodes and edges and 2^n Hamiltonian cycles with a one-to-one correspondence with 2^n truth assignments.
 - 2. Add nodes to impose constraints arising from clauses.
 - 3. Construction takes O(nk) time.
- G contains n paths $P_1, P_2, \ldots P_n$, one for each variable.
- ► Each P_i contains b = 3k + 3 nodes v_{i,1}, v_{i,2}, ... v_{i,b}, three for each clause and some extra nodes.

3-SAT \leq_P Hamiltonian Cycle: Constructing G



3-SAT \leq_P Hamiltonian Cycle: Modelling clauses



3-SAT \leq_P Hamiltonian Cycle: Modelling clauses



Strategy Sequencing Problems Partitioning Problems Other Problems Example • Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. s P_1 P_2 t


Example • Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. c_1 P_1 P_2 t Nodes for c_2 Nodes for c_1

Partitioning Problems

Sequencing Problems

Strategy

Other Problems

Strategy Sequencing Problems Partitioning Problems Other Problems Example • Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. c_2 S c_1 P_1 P_2

t

Strategy Sequencing Problems Partitioning Problems Other Problems Example • Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. c_2 s c_1 P_1 P_2

t

Strategy Sequencing Problems Partitioning Problems Other Problems Example • Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. c_2 s c_1 P_1 P_2

t



▶ 3-SAT instance is satisfiable \rightarrow *G* has a Hamiltonian cycle.



- ▶ 3-SAT instance is satisfiable \rightarrow *G* has a Hamiltonian cycle.
 - ► Construct a Hamiltonian cycle C as follows:
 - If $x_i = 1$, traverse P_i from left to right in C.
 - Otherwise, traverse P_i from right to left in C.
 - For each clause C_j, there is at least one term set to 1. If the term is x_i, splice c_j into C using edge from v_{i,3j} and edge to v_{i,3j+1}. Analogous construction if term is x_i.



- G has a Hamiltonian cycle $C \rightarrow 3$ -SAT instance is satisfiable.
 - ▶ If C enters c_j on an edge from $v_{i,3j}$, it must leave c_j along the edge to $v_{i,3j+1}$.
 - Analogous statement if C enters c_j on an edge from $v_{i,3j+1}$.



- G has a Hamiltonian cycle $C \rightarrow 3$ -SAT instance is satisfiable.
 - ▶ If C enters c_j on an edge from $v_{i,3j}$, it must leave c_j along the edge to $v_{i,3j+1}$.
 - Analogous statement if C enters c_j on an edge from $v_{i,3j+1}$.
 - ▶ Nodes immediately before and after *c_j* in *C* are themselves connected by an edge *e* in *G*.



- G has a Hamiltonian cycle $C \rightarrow 3$ -SAT instance is satisfiable.
 - ▶ If C enters c_j on an edge from $v_{i,3j}$, it must leave c_j along the edge to $v_{i,3j+1}$.
 - Analogous statement if C enters c_j on an edge from $v_{i,3j+1}$.
 - ▶ Nodes immediately before and after *c_j* in *C* are themselves connected by an edge *e* in *G*.
 - If we remove all such edges e from C, we get a Hamiltonian cycle C' in $G \{c_1, c_2, \ldots, c_k\}$.
 - Use C' to construct truth assignment to variables; prove assignment is satisfying

April 28, 30, 2015

The Traveling Salesman Problem

- A salesman must visit *n* cities v_1, v_2, \ldots, v_n starting at home city v_1 .
- Salesman must find a *tour*, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.

The Traveling Salesman Problem

- A salesman must visit *n* cities v_1, v_2, \ldots, v_n starting at home city v_1 .
- Salesman must find a *tour*, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.
- For every pair of cities v_i and v_j , $d(v_i, v_j) > 0$ is the distance from v_i to v_j .
- A tour is a permutation $v_{i_1} = v_1, v_{i_2}, \ldots v_{i_n}$.
- The *length* of the tour is $\sum_{j=1}^{n-1} d(v_{i_j}v_{i_{j+1}}) + d(v_{i_n}, v_{i_1})$.

The Traveling Salesman Problem

- A salesman must visit *n* cities v_1, v_2, \ldots, v_n starting at home city v_1 .
- Salesman must find a *tour*, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.
- For every pair of cities v_i and v_j , $d(v_i, v_j) > 0$ is the distance from v_i to v_j .
- A tour is a permutation $v_{i_1} = v_1, v_{i_2}, \ldots v_{i_n}$.
- The *length* of the tour is $\sum_{j=1}^{n-1} d(v_{i_j}v_{i_{j+1}}) + d(v_{i_n}, v_{i_1})$.

TRAVELLING SALESMAN

INSTANCE: A set V of n cities, a function $d: V \times V \rightarrow \mathbb{R}^+$, and a number D > 0.

QUESTION: Is there a tour of length at most *D*?



(1977) 120 cities, Groetschel Images taken from http://tsp.gatech.edu

T. M. Murali

April 28, 30, 2015

NP-Complete Problems



(1987) 532 AT&T switch locations, Padberg and Rinaldi Images taken from http://tsp.gatech.edu



(1987) 13,509 cities with population \geq 500, Applegate, Bixby, Chváthal, and Cook Images taken from http://tsp.gatech.edu



(2001) 15,112 cities, Applegate, Bixby, Chváthal, and Cook Images taken from http://tsp.gatech.edu



(2004) 24978, cities, Applegate, Bixby, Chváthal, Cook, and Helsgaum Images taken from http://tsp.gatech.edu

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?
- ▶ Claim: HAMILTONIAN CYCLE \leq_P TRAVELLING SALESMAN.

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?
- ▶ Claim: HAMILTONIAN CYCLE \leq_P TRAVELLING SALESMAN.

HAMILTONIAN CYCLE	TRAVELLING SALESMAN
Directed graph $G(V, E)$	Cities
Edges have identical weights	Distances between cities can vary
Not all pairs of nodes are connected in G	Every pair of cities has a distance
(u, v) and (v, u) may both be edges	$d(v_i, v_j) eq d(v_j, v_i)$, in general
Does a cycle exist?	Does a tour of length $\leq D$ exist?

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?
- Claim: HAMILTONIAN CYCLE \leq_P TRAVELLING SALESMAN.

HAMILTONIAN CYCLE	TRAVELLING SALESMAN
Directed graph $G(V, E)$	Cities
Edges have identical weights	Distances between cities can vary
Not all pairs of nodes are connected in G	Every pair of cities has a distance
(u, v) and (v, u) may both be edges	$d(v_i, v_j) eq d(v_j, v_i)$, in general
Does a cycle exist?	Does a tour of length $\leq D$ exist?

- Given a directed graph G(V, E) (instance of HAMILTONIAN CYCLE),
 - Create a city v_i for each node $i \in V$.
 - Define $d(v_i, v_j) = 1$ if $(i, j) \in E$.
 - Define $d(v_i, v_j) = 2$ if $(i, j) \notin E$.

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?
- Claim: HAMILTONIAN CYCLE \leq_P TRAVELLING SALESMAN.

HAMILTONIAN CYCLE	TRAVELLING SALESMAN
Directed graph $G(V, E)$	Cities
Edges have identical weights	Distances between cities can vary
Not all pairs of nodes are connected in G	Every pair of cities has a distance
(u, v) and (v, u) may both be edges	$d(v_i, v_j) eq d(v_j, v_i)$, in general
Does a cycle exist?	Does a tour of length $\leq D$ exist?

- Given a directed graph G(V, E) (instance of HAMILTONIAN CYCLE),
 - Create a city v_i for each node $i \in V$.
 - Define $d(v_i, v_j) = 1$ if $(i, j) \in E$.
 - Define $d(v_i, v_j) = 2$ if $(i, j) \notin E$.
- Claim: G has a Hamiltonian cycle iff the instance of Travelling Salesman has a tour of length at most

- Why is the problem in \mathcal{NP} ?
- Why is the problem \mathcal{NP} -Complete?
- Claim: HAMILTONIAN CYCLE \leq_P TRAVELLING SALESMAN.

HAMILTONIAN CYCLE	TRAVELLING SALESMAN
Directed graph $G(V, E)$	Cities
Edges have identical weights	Distances between cities can vary
Not all pairs of nodes are connected in G	Every pair of cities has a distance
(u, v) and (v, u) may both be edges	$d(v_i, v_j) eq d(v_j, v_i)$, in general
Does a cycle exist?	Does a tour of length $\leq D$ exist?

- Given a directed graph G(V, E) (instance of HAMILTONIAN CYCLE),
 - Create a city v_i for each node $i \in V$.
 - Define $d(v_i, v_j) = 1$ if $(i, j) \in E$.
 - Define $d(v_i, v_j) = 2$ if $(i, j) \notin E$.
- Claim: G has a Hamiltonian cycle iff the instance of Travelling Salesman has a tour of length at most n.

Special Cases and Extensions that are \mathcal{NP} -Complete

- ► HAMILTONIAN CYCLE for undirected graphs.
- ► HAMILTONIAN PATH for directed and undirected graphs.
- ► TRAVELLING SALESMAN with symmetric distances (by reducing HAMILTONIAN CYCLE for undirected graphs to it).
- \blacktriangleright TRAVELLING SALESMAN with distances defined by points on the plane.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
	3	B-Dimensional	Matching	

BIPARTITE MATCHING

INSTANCE: Disjoint sets X, Y, each of size n, and a set $T \subseteq X \times Y$ of pairs

QUESTION: Is there a set of *n* pairs in *T* such that each element of $X \cup Y$ is contained in exactly one of these pairs?

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
	3	-Dimensional	Matching	
		0	Ĉ	

BIPARTITE MATCHING

INSTANCE: Disjoint sets X, Y, each of size n, and a set $T \subseteq X \times Y$ of pairs

QUESTION: Is there a set of *n* pairs in *T* such that each element of $X \cup Y$ is contained in exactly one of these pairs?



▶ 3-DIMENSIONAL MATCHING is a harder version of BIPARTITE MATCHING. BIPARTITE MATCHING **INSTANCE:** Disjoint sets X, Y, each of size n, and a set $T \subseteq X \times Y$ of pairs **QUESTION:** Is there a set of n pairs in T such that each element of $X \cup Y$ is contained in exactly one of these pairs?



▶ 3-DIMENSIONAL MATCHING is a harder version of BIPARTITE MATCHING. 3-DIMENSIONAL MATCHING **INSTANCE:** Disjoint sets X, Y, and Z, each of size n, and a set $T \subseteq X \times Y \times Z$ of triples **QUESTION:** Is there a set of n triples in T such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?



- ▶ 3-DIMENSIONAL MATCHING is a harder version of BIPARTITE MATCHING. 3-DIMENSIONAL MATCHING **INSTANCE:** Disjoint sets X, Y, and Z, each of size n, and a set $T \subseteq X \times Y \times Z$ of triples **QUESTION:** Is there a set of n triples in T such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?
- ► Easy to show 3-DIMENSIONAL MATCHING \leq_P SET COVER and 3-DIMENSIONAL MATCHING \leq_P SET PACKING.

3-Dimensional Matching is \mathcal{NP} -Complete

• Why is the problem in \mathcal{NP} ?

3-Dimensional Matching is \mathcal{NP} -Complete

- Why is the problem in \mathcal{NP} ?
- ▶ Show that 3-SAT \leq_P 3-DIMENSIONAL MATCHING. → Jump to Colouring
- Strategy:
 - Start with an instance of 3-SAT with n variables and k clauses.
 - Create a gadget for each variable x_i that encodes the choice of truth assignment to x_i.
 - Add gadgets that encode constraints imposed by clauses.

3-SAT \leq_P **3-Dimensional Matching: Variables**



Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.

- Each x_i corresponds to a variable gadget i with 2k core elements
 A_i = {a_{i.1}, a_{i.2}, ... a_{i.2k}} and 2k tips
 - $B_i = \{b_{i,1}, b_{i,2}, \dots b_{i,2k}\}$ and 2k the $B_i = \{b_{i,1}, b_{i,2}, \dots b_{i,2k}\}.$
- For each 1 ≤ j ≤ 2k, variable gadget i includes a triple t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j}).
- ► A triple (tip) is even if j is even. Otherwise, the triple (tip) is odd.
- Only these triples contain elements in A_i.

3-SAT \leq_P **3-Dimensional Matching: Variables**



- Each x_i corresponds to a variable gadget i with 2k core elements
 A_i = {a_{i,1}, a_{i,2}, ... a_{i,2k}} and 2k tips
 - $B_i = \{b_{i,1}, b_{i,2}, \dots b_{i,2k}\}.$
- For each 1 ≤ j ≤ 2k, variable gadget i includes a triple t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j}).
- ► A triple (tip) is even if j is even. Otherwise, the triple (tip) is odd.
- Only these triples contain elements in A_i.

• In any perfect matching, we can cover the elements in A_i

3-SAT \leq_P **3-Dimensional Matching: Variables**



- Each x_i corresponds to a variable gadget i with 2k core elements
 - $A_i = \{a_{i,1}, a_{i,2}, \dots a_{i,2k}\} \text{ and } 2k \text{ tips} \\ B_i = \{b_{i,1}, b_{i,2}, \dots b_{i,2k}\}.$
- For each 1 ≤ j ≤ 2k, variable gadget i includes a triple t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j}).
- ► A triple (tip) is even if j is even. Otherwise, the triple (tip) is odd.
- ► Only these triples contain elements in A_i.
- In any perfect matching, we can cover the elements in A_i either using all the even triples in gadget i or all the odd triples in the gadget.
- ► Even triples used, odd tips free ≡ x_i = 0; odd triples used, even tips free ≡ x_i = 1.

3-SAT \leq_P **3-Dimensional Matching: Clauses**

• Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.



Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT \leq_P **3-Dimensional Matching: Clauses**



Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.

• Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.

C₁ says "The matching on the cores of the gadgets should leave the even tips of gadget 1 free; or it should leave the odd tips of gadget 2 free; or it should leave the even tips of gadget 3 free."

3-SAT \leq_P **3-Dimensional Matching: Clauses**



Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.

• Consider the clause $C_1 = x_1 \vee \overline{x_2} \vee x_3$.

- C₁ says "The matching on the cores of the gadgets should leave the even tips of gadget 1 free; or it should leave the odd tips of gadget 2 free; or it should leave the even tips of gadget 3 free."
- Clause gadget j for clause C_j contains two core elements P_j = {p_j, p'_j} and three triples:
 - C_j contains x_i : add triple $(p_j, p'_j, b_{i,2j})$.
 - ► C_j contains x_i: add triple (p_j, p'_j, b_{i,2j-1}).





• Satisfying assignment \rightarrow matching.

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!
 - Add (n − 1)k cleanup gadgets to allow the remaining (n − 1)k tips to be covered: cleanup gadget i contains two core elements Q = {q_i, q_i'} and triple (q_i, q_i', b) for every tip b in variable gadget i.

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!
 - Add (n − 1)k cleanup gadgets to allow the remaining (n − 1)k tips to be covered: cleanup gadget i contains two core elements Q = {q_i, q_i'} and triple (q_i, q_i', b) for every tip b in variable gadget i.
- Matching \rightarrow satisfying assignment.

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!
 - Add (n − 1)k cleanup gadgets to allow the remaining (n − 1)k tips to be covered: cleanup gadget i contains two core elements Q = {q_i, q_i'} and triple (q_i, q_i', b) for every tip b in variable gadget i.
- Matching \rightarrow satisfying assignment.
 - Matching chooses all even a_{ij} ($x_i = 0$) or all odd a_{ij} ($x_i = 1$).

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!
 - Add (n − 1)k cleanup gadgets to allow the remaining (n − 1)k tips to be covered: cleanup gadget i contains two core elements Q = {q_i, q_i'} and triple (q_i, q_i', b) for every tip b in variable gadget i.
- Matching \rightarrow satisfying assignment.
 - Matching chooses all even a_{ij} ($x_i = 0$) or all odd a_{ij} ($x_i = 1$).
 - Is clause C_j satisfied?

- Satisfying assignment \rightarrow matching.
 - Make appropriate choices for the core of each variable gadget.
 - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
 - We have not covered all the tips!
 - Add (n − 1)k cleanup gadgets to allow the remaining (n − 1)k tips to be covered: cleanup gadget i contains two core elements Q = {q_i, q_i'} and triple (q_i, q_i', b) for every tip b in variable gadget i.
- Matching \rightarrow satisfying assignment.
 - Matching chooses all even a_{ij} ($x_i = 0$) or all odd a_{ij} ($x_i = 1$).
 - ► Is clause C_j satisfied? Core in clause gadget j is covered by some triple ⇒ other element in the triple must be a tip element from the correct odd/even set in the three variable gadgets corresponding to a term in C_j.

► Did we create an instance of 3-DIMENSIONAL MATCHING?

- ▶ Did we create an instance of 3-DIMENSIONAL MATCHING?
- We need three sets X, Y, and Z of equal size.

- ► Did we create an instance of 3-DIMENSIONAL MATCHING?
- We need three sets X, Y, and Z of equal size.
- How many elements do we have?
 - 2nk a_{ij} elements.
 - 2nk b_{ij} elements.
 - ▶ *k p_j* elements.
 - $k p'_i$ elements.
 - $(n-1)k q_i$ elements.
 - $(n-1)k q'_i$ elements.

- ► Did we create an instance of 3-DIMENSIONAL MATCHING?
- We need three sets X, Y, and Z of equal size.
- How many elements do we have?
 - 2nk a_{ij} elements.
 - 2nk b_{ij} elements.
 - ▶ *k p_j* elements.
 - ▶ k p'_i elements.
 - $(n-1)k q_i$ elements.
 - $(n-1)k q'_i$ elements.
- X is the union of a_{ij} with even j, the set of all p_j and the set of all q_i .
- Y is the union of a_{ij} with odd j, the set if all p'_i and the set of all q'_i .
- ► Z is the set of all b_{ij}.

- ► Did we create an instance of 3-DIMENSIONAL MATCHING?
- We need three sets X, Y, and Z of equal size.
- How many elements do we have?
 - 2nk a_{ij} elements.
 - 2nk b_{ij} elements.
 - ▶ *k p_j* elements.
 - $k p'_i$ elements.
 - $(n-1)k q_i$ elements.
 - $(n-1)k q'_i$ elements.
- X is the union of a_{ij} with even j, the set of all p_j and the set of all q_i .
- Y is the union of a_{ij} with odd j, the set if all p'_i and the set of all q'_i .
- Z is the set of all b_{ij} .
- ► Each triple contains exactly one element from *X*, *Y*, and *Z*.

Colouring maps



Colouring maps



► Any map can be coloured with four colours (Appel and Hakken, 1976).



• Given an undirected graph G(V, E), a *k*-colouring of G is a function $f: V \to \{1, 2, ..., k\}$ such that for every edge $(u, v) \in E$, $f(u) \neq f(v)$.



Applications of Graph Colouring

- 1. Job scheduling: assign jobs to *n* processors under constraints that certain pairs of jobs cannot be scheduled at the same time.
- 2. Compiler design: assign variables to k registers but two variables being used at the same time cannot be assigned to the same register.
- 3. Wavelength assignment: assign one of *k* transmitting wavelengths to each of *n* wireless devices. If two devices are close to each other, they must get different wavelengths.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems		
2-Colouring						

► How hard is 2-COLOURING?

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
		2-Colour	ing	

- ▶ How hard is 2-COLOURING?
- ▶ Claim: A graph is 2-colourable if and only if it is bipartite.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
		2-Colour	ing	
► How	' hard is 2-COL	OURING?		

- Claim: A graph is 2-colourable if and only if it is bipartite.
- Testing 2-colourability is possible in O(|V| + |E|) time.

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
		2-Colour	ing	

- ▶ How hard is 2-COLOURING?
- Claim: A graph is 2-colourable if and only if it is bipartite.
- Testing 2-colourability is possible in O(|V| + |E|) time.
- ▶ What about 3-COLOURING? Is it easy to exhibit a certificate that a graph *cannot* be coloured with three colours?



Figure 8.10 A graph that is not 3-colorable.

3-Colouring is $\mathcal{NP}\text{-}\text{Complete}$

• Why is 3-Colouring in \mathcal{NP} ?

3-Colouring is \mathcal{NP} -Complete

- Why is 3-Colouring in \mathcal{NP} ?
- ▶ 3-SAT \leq_P 3-Colouring.

3-SAT \leq_P **3-Colouring: Encoding Variables**



Figure 8.11 The beginning of the reduction for 3-Coloring.

► x_i corresponds to node v_i and x_i corresponds to node v_i.

3-SAT \leq_P **3-Colouring: Encoding Variables**



Figure 8.11 The beginning of the reduction for 3-Coloring.

- x_i corresponds to node v_i and x̄_i corresponds to node v̄_i.
- ► In any 3-Colouring, nodes v_i and v_i get a colour different from Base.
- True colour: colour assigned to the True node; False colour: colour assigned to the False node.
- Set x_i to 1 iff v_i gets the *True* colour.

• Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.



Figure 8.12 Attaching a subgraph to represent the clause $x_1 \lor \overline{x}_2 \lor x_3$.

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.
- Attach a six-node subgraph for this clause to the rest of the graph.



Figure 8.12 Attaching a subgraph to represent the clause $x_1 \lor \overline{x}_2 \lor x_3$.

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.
- Attach a six-node subgraph for this clause to the rest of the graph.
- Claim: Top node in the subgraph can be coloured in a 3-colouring iff one of v₁, v₂, or v₃ does not get the *False* colour.



Figure 8.12 Attaching a subgraph to represent the clause $x_1 \lor \overline{x}_2 \lor x_3$.

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3.$
- Attach a six-node subgraph for this clause to the rest of the graph.
- Claim: Top node in the subgraph can be coloured in a 3-colouring iff one of v₁, v₂, or v₃ does not get the False colour.
- Claim: Graph is
 3-colourable iff instance of
 3-SAT is satisfiable.

Subset Sum

SUBSET SUM **INSTANCE:** A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems			
Subset Sum							
SUBS	Set Sum						

- **INSTANCE:** A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?
- SUBSET SUM is a special case of the KNAPSACK PROBLEM (see Chapter 6.4 of the textbook).

Strategy	3-SAT	Sequencing Problems	Partitioning Problems	Other Problems
		Subset S	um	
SUBS	SET SUM			

INSTANCE: A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?

- SUBSET SUM is a special case of the KNAPSACK PROBLEM (see Chapter 6.4 of the textbook).
- ► There is a dynamic programming algorithm for SUBSET SUM that runs in O(nW) time.
| Strategy | 3-SAT | Sequencing Problems | Partitioning Problems | Other Problems |
|----------|---------|---------------------|-----------------------|----------------|
| | | Subset S | | |
| | | Subset S | um | |
| SUBS | SET SUM | | | |

INSTANCE: A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?

- SUBSET SUM is a special case of the KNAPSACK PROBLEM (see Chapter 6.4 of the textbook).
- ► There is a dynamic programming algorithm for SUBSET SUM that runs in O(nW) time. This algorithm's running time is exponential in the size of the input.

Subset Sum

INSTANCE: A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?

- SUBSET SUM is a special case of the KNAPSACK PROBLEM (see Chapter 6.4 of the textbook).
- ► There is a dynamic programming algorithm for SUBSET SUM that runs in O(nW) time. This algorithm's running time is exponential in the size of the input.
- ▶ Claim: SUBSET SUM is NP-Complete,
 3-DIMENSIONAL MATCHING ≤_P SUBSET SUM.

Subset Sum

INSTANCE: A set of *n* natural numbers w_1, w_2, \ldots, w_n and a target *W*. **QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is *W*?

- SUBSET SUM is a special case of the KNAPSACK PROBLEM (see Chapter 6.4 of the textbook).
- ► There is a dynamic programming algorithm for SUBSET SUM that runs in O(nW) time. This algorithm's running time is exponential in the size of the input.
- ▶ Claim: SUBSET SUM is NP-Complete,
 3-DIMENSIONAL MATCHING ≤_P SUBSET SUM.
- ► Caveat: Special case of SUBSET SUM in which *W* is bounded by a polynomial function of *n* is not *NP*-Complete (read pages 494–495 of your textbook).

Examples of Hard Computational Problems

(from Kevin Wayne's slides at Princeton University)

- ► Aerospace engineering: optimal mesh partitioning for finite elements.
- Biology: protein folding.
- Chemical engineering: heat exchanger network synthesis.
- Civil engineering: equilibrium of urban traffic flow.
- Economics: computation of arbitrage in financial markets with friction.
- Electrical engineering: VLSI layout.
- > Environmental engineering: optimal placement of contaminant sensors.
- > Financial engineering: find minimum risk portfolio of given return.
- ► Game theory: find Nash equilibrium that maximizes social welfare.
- Genomics: phylogeny reconstruction.
- ▶ Mechanical engineering: structure of turbulence in sheared flows.
- ▶ Medicine: reconstructing 3-D shape from biplane angiocardiogram.
- Operations research: optimal resource allocation.
- Physics: partition function of 3-D Ising model in statistical mechanics.
- Politics: Shapley-Shubik voting power.
- Pop culture: Minesweeper consistency.
- Statistics: optimal experimental design.