NP-Completeness Proofs

Matt Williamson[1]

[1]Lane Department of Computer Science and Electrical Engineering
West Virginia University

Graph Theory, Packing, and Covering

Outline

Outline

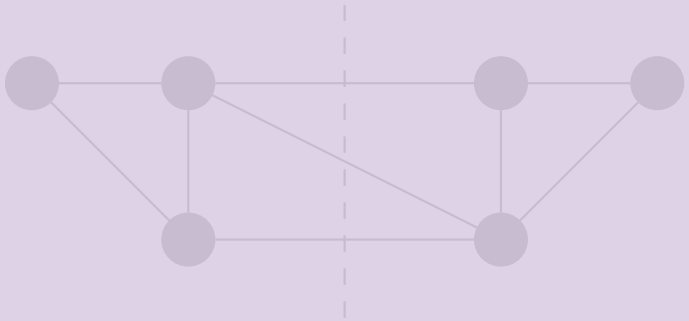
Outline

Max Bisection

Problem

Given a graph $G = (V, E)$, we are looking for a cut S, $V - S$ of size K or more such that $|S| = |V - S|$. Note that if $|V| = n$ is odd, then the problem is trivial.

Example

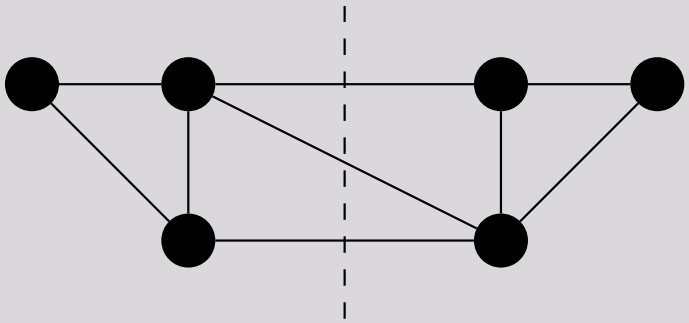


Max Bisection

Problem

Given a graph $G = (V, E)$, we are looking for a cut S, $V - S$ of size K or more such that $|S| = |V - S|$. Note that if $|V| = n$ is odd, then the problem is trivial.

Example



MAX BISECTION (contd.)

Lemma

MAX BISECTION *is NP-complete.*

Proof.

We modify a given instance of Max Cut so that the constraint $|S| = |V - S|$ is easy to satisfy. This gives us a solution if and only if the instance of MAX CUT does. The trick is to add $|V|$ completely disconnected nodes to G. We can make a bisection and add the new nodes between the two sections to get $|S| = |V - S|$ to get the result. □

MAX BISECTION (contd.)

Lemma

MAX BISECTION *is NP-complete.*

Proof.

We modify a given instance of Max Cut so that the constraint $|S| = |V - S|$ is easy to satisfy. This gives us a solution if and only if the instance of MAX CUT does. The trick is to add $|V|$ completely disconnected nodes to G. We can make a bisection and add the new nodes between the two sections to get $|S| = |V - S|$ to get the result. □

MAX BISECTION (contd.)

Lemma

MAX BISECTION *is NP-complete.*

Proof.

We modify a given instance of Max Cut so that the constraint $|S| = |V - S|$ is easy to satisfy. This gives us a solution if and only if the instance of MAX CUT does. The trick is to add $|V|$ completely disconnected nodes to G. We can make a bisection and add the new nodes between the two sections to get $|S| = |V - S|$ to get the result. □

Bisection Width

Problem

This is the same problem as Max Bisection except we want a bisection size at most K. We can say this would be the minimum bisection.

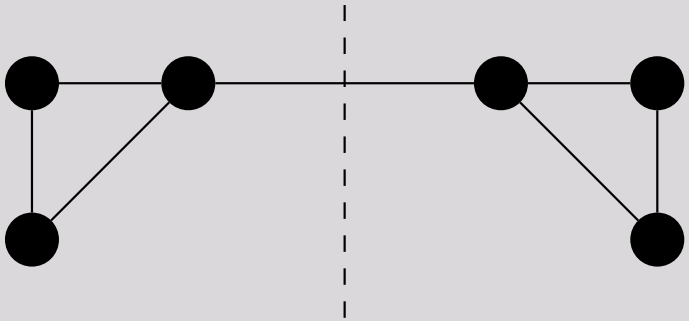
Example

Bisection Width

Problem

This is the same problem as Max Bisection except we want a bisection size at most K. We can say this would be the minimum bisection.

Example



Bisection Width (contd.)

Theorem

Bisection Width *is NP-complete.*

Proof.

Given a graph $G = (V, E)$ where $|V| = 2n$ is an even number, we can see that we have a bisection of size K or more if and only if the complement of G has a bisection of size $n^2 - K$. □

BISECTION WIDTH (contd.)

Theorem

BISECTION WIDTH *is NP-complete.*

Proof.

Given a graph $G = (V, E)$ where $|V| = 2n$ is an even number, we can see that we have a bisection of size K or more if and only if the complement of G has a bisection of size $n^2 - K$. □

Outline

Hamilton Path

Problem

Given a graph $G = (V, E)$ and an undirected path, does it have a Hamilton path, a path visiting each node exactly once?

Theorem

Hamilton Path *is NP-complete.*

Proof

We reduce 3SAT to this problem. We will first need to express the properties of 3SAT as graph elements.

Hamilton Path

Problem

Given a graph $G = (V, E)$ and an undirected path, does it have a Hamilton path, a path visiting each node exactly once?

Theorem

Hamilton Path *is NP-complete.*

Proof

We reduce 3SAT to this problem. We will first need to express the properties of 3SAT as graph elements.

Hamilton Path

Problem

Given a graph $G = (V, E)$ and an undirected path, does it have a Hamilton path, a path visiting each node exactly once?

Theorem

Hamilton Path *is NP-complete.*

Proof

We reduce 3SAT to this problem. We will first need to express the properties of 3SAT as graph elements.

Hamilton Path

Problem

Given a graph $G = (V, E)$ and an undirected path, does it have a Hamilton path, a path visiting each node exactly once?

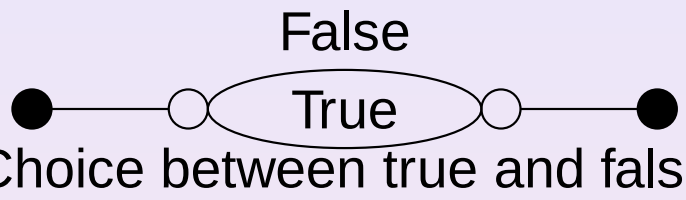
Theorem

Hamilton Path *is NP-complete.*

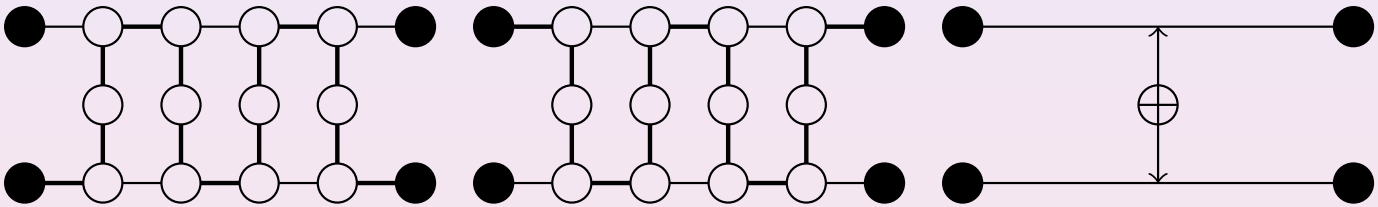
Proof

We reduce 3SAT to this problem. We will first need to express the properties of 3SAT as graph elements.

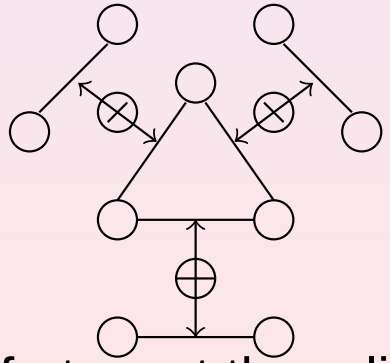
Hamilton Path (contd.)



Choice between true and false



Consistency: all x and $\neg x$ have opposite values, can use xor gadget



Restriction of at most three literals/clause

Hamilton Path (contd.)

Observation

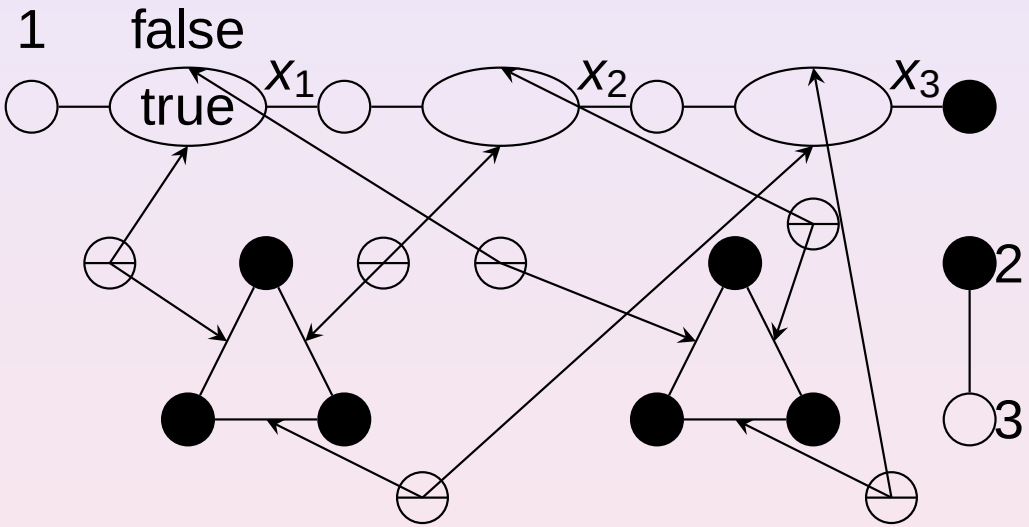
Suppose we have made sure that each side of a triangle is traversed by a Hamilton path if and only if the corresponding literal is false. Then at least one literal is true, since otherwise all three sides are traversed and there is no Hamilton path.

Hamilton Path (contd.)

Observation

Suppose we have made sure that each side of a triangle is traversed by a Hamilton path if and only if the corresponding literal is false. Then at least one literal is true, since otherwise all three sides are traversed and there is no Hamilton path.

Hamilton Path (contd.)



Example: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ gives graph where black nodes make up large clique and are connected

Hamilton Path (contd.)

Proof.

We can get a Hamilton path if and only if we have a satisfying truth assignment. Starting at node 1, one of the two parallel edges for each choice must be traversed. The nodes connected by the xor gadgets will also be traversed. Since the black nodes are a large clique, we can connect the remaining black nodes and traverse to nodes 2 and 3. This path provide a satisfying truth assignment.

Given a satisfying truth assignment, we can create a Hamilton path starting at node 1. □

Hamilton Path (contd.)

Proof.

We can get a Hamilton path if and only if we have a satisfying truth assignment. Starting at node 1, one of the two parallel edges for each choice must be traversed. The nodes connected by the xor gadgets will also be traversed. Since the black nodes are a large clique, we can connect the remaining black nodes and traverse to nodes 2 and 3. This path provide a satisfying truth assignment.

Given a satisfying truth assignment, we can create a Hamilton path starting at node 1. □

Hamilton Path (contd.)

Proof.

We can get a Hamilton path if and only if we have a satisfying truth assignment. Starting at node 1, one of the two parallel edges for each choice must be traversed. The nodes connected by the xor gadgets will also be traversed. Since the black nodes are a large clique, we can connect the remaining black nodes and traverse to nodes 2 and 3. This path provide a satisfying truth assignment.

Given a satisfying truth assignment, we can create a Hamilton path starting at node 1. □

Hamilton Path (contd.)

Proof.

We can get a Hamilton path if and only if we have a satisfying truth assignment. Starting at node 1, one of the two parallel edges for each choice must be traversed. The nodes connected by the xor gadgets will also be traversed. Since the black nodes are a large clique, we can connect the remaining black nodes and traverse to nodes 2 and 3. This path provide a satisfying truth assignment.

Given a satisfying truth assignment, we can create a Hamilton path starting at node 1. □

Hamilton Path (contd.)

Proof.

We can get a Hamilton path if and only if we have a satisfying truth assignment. Starting at node 1, one of the two parallel edges for each choice must be traversed. The nodes connected by the xor gadgets will also be traversed. Since the black nodes are a large clique, we can connect the remaining black nodes and traverse to nodes 2 and 3. This path provide a satisfying truth assignment.

Given a satisfying truth assignment, we can create a Hamilton path starting at node 1. □

Hamilton Circuit

Problem

Given an undirected graph $G = (V, E)$, can we create a Hamilton circuit, a cycle visiting each node exactly once and returning to the starting node?

Corollary

Hamilton Circuit *is NP-complete.*

Hamilton Circuit

Problem

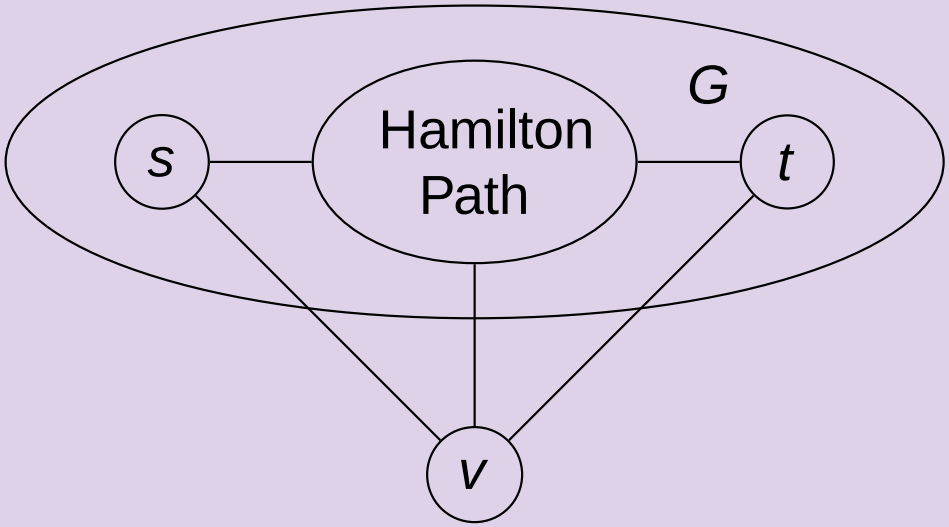
Given an undirected graph $G = (V, E)$, can we create a Hamilton circuit, a cycle visiting each node exactly once and returning to the starting node?

Corollary

Hamilton Circuit *is NP-complete.*

HAMILTON CIRCUIT (contd.)

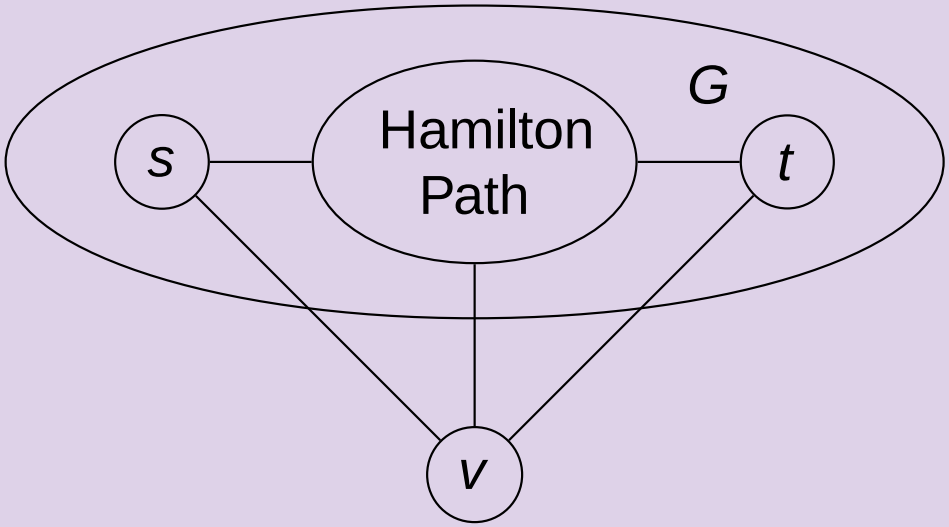
Proof.



Given an instance of graph G containing a Hamilton path, we add a new node v that is connected to all other nodes in G. We simply connect the last node of the Hamilton path to v, and then connect to the first node in the path, creating a Hamilton circuit. □

Hamilton Circuit (contd.)

Proof.



Given an instance of graph G containing a Hamilton path, we add a new node v that is connected to all other nodes in G. We simply connect the last node of the Hamilton path to v, and then connect to the first node in the path, creating a Hamilton circuit. □

Outline

Longest Path

Problem

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple (visiting each vertex no more than once) path with K or more edges?

Corollary

Longest Path *is NP-complete.*

Proof.

We reduce from Hamilton Path. Given an instance of Hamilton Path in graph G', let $K = |V| - 1$. We have a simple path of length K if and only if G' contains a Hamilton path. □

Longest Path

Problem

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple (visiting each vertex no more than once) path with K or more edges?

Corollary

Longest Path *is NP-complete.*

Proof.

We reduce from Hamilton Path. Given an instance of Hamilton Path in graph G', let $K = |V| - 1$. We have a simple path of length K if and only if G' contains a Hamilton path. □

Longest Path

Problem

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple (visiting each vertex no more than once) path with K or more edges?

Corollary

Longest Path *is NP-complete.*

Proof.

We reduce from Hamilton Path. Given an instance of Hamilton Path in graph G', let $K = |V| - 1$. We have a simple path of length K if and only if G' contains a Hamilton path. □

Longest Path

Problem

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple (visiting each vertex no more than once) path with K or more edges?

Corollary

Longest Path *is NP-complete.*

Proof.

We reduce from Hamilton Path. Given an instance of Hamilton Path in graph G', let $K = |V| - 1$. We have a simple path of length K if and only if G' contains a Hamilton path. □

Longest Path

Problem

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple (visiting each vertex no more than once) path with K or more edges?

Corollary

Longest Path *is NP-complete.*

Proof.

We reduce from Hamilton Path. Given an instance of Hamilton Path in graph G', let $K = |V| - 1$. We have a simple path of length K if and only if G' contains a Hamilton path. □

Longest Circuit

Longest Circuit

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple circuit, a circuit visiting each vertex no more than once, with K or more edges?

Corollary

Longest Circuit *is NP-complete.*

Proof.

We reduce from Hamilton Circuit. Given an instance of Hamilton Circuit, if the graph contains a Hamilton circuit, let $K = |V|$, where $|V|$ is the number of nodes in the path. Then we have a simple circuit of length K if and only if the graph contains a Hamilton circuit. □

Longest Circuit

Longest Circuit

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple circuit, a circuit visiting each vertex no more than once, with K or more edges?

Corollary

Longest Circuit *is NP-complete.*

Proof.

We reduce from Hamilton Circuit. Given an instance of Hamilton Circuit, if the graph contains a Hamilton circuit, let $K = |V|$, where $|V|$ is the number of nodes in the path. Then we have a simple circuit of length K if and only if the graph contains a Hamilton circuit. □

Longest Circuit

Longest Circuit

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple circuit, a circuit visiting each vertex no more than once, with K or more edges?

Corollary

Longest Circuit *is NP-complete.*

Proof.

We reduce from Hamilton Circuit. Given an instance of Hamilton Circuit, if the graph contains a Hamilton circuit, let $K = |V|$, where $|V|$ is the number of nodes in the path. Then we have a simple circuit of length K if and only if the graph contains a Hamilton circuit. □

Longest Circuit

Longest Circuit

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple circuit, a circuit visiting each vertex no more than once, with K or more edges?

Corollary

Longest Circuit *is NP-complete.*

Proof.

We reduce from Hamilton Circuit. Given an instance of Hamilton Circuit, if the graph contains a Hamilton circuit, let $K = |V|$, where $|V|$ is the number of nodes in the path. Then we have a simple circuit of length K if and only if the graph contains a Hamilton circuit. □

Longest Circuit

Longest Circuit

Given an undirected graph $G = (V, E)$ and a positive integer $K \leq |V|$, does G contain a simple circuit, a circuit visiting each vertex no more than once, with K or more edges?

Corollary

Longest Circuit *is NP-complete.*

Proof.

We reduce from Hamilton Circuit. Given an instance of Hamilton Circuit, if the graph contains a Hamilton circuit, let $K = |V|$, where $|V|$ is the number of nodes in the path. Then we have a simple circuit of length K if and only if the graph contains a Hamilton circuit. □

Outline

TSP (D)

Problem

This is the decision version of TSP. Given n cities 1, …, n, a nonnegative integer distance d_{ij} between any two cities i and j, and an integer bound B, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most B?

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$, a nonnegative integer distance d_{ij} between any two cities i and j, and an integer bound B, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most B?

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$.

We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$, a nonnegative integer distance d_{ij} between any two cities i and j, and an integer bound B, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most B?

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

TSP (D)

Problem

This is the decision version of TSP. Given n cities $1, \ldots, n$*, a nonnegative integer distance* d_{ij} *between any two cities* i *and* j*, and an integer bound* B*, can we find a tour (visiting each city exactly once and returning to the starting city) of length at most* B*?*

Corollary

TSP (D) *is NP-complete.*

Proof.

We reduce HAMILTON CIRCUIT to this problem. Given an instance of a graph G, we create G' as follows: We first make G' complete. We let $d_{ij} = 0$ if edge (i, j) is in G. Otherwise, we let $d_{ij} = 1$. We solve for TSP (D). If the cost of the tour is 0, then we have a Hamilton circuit. □

Outline

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING

Problem

We want to color the vertices of a graph with three colors such that no two adjacent nodes have the same color.

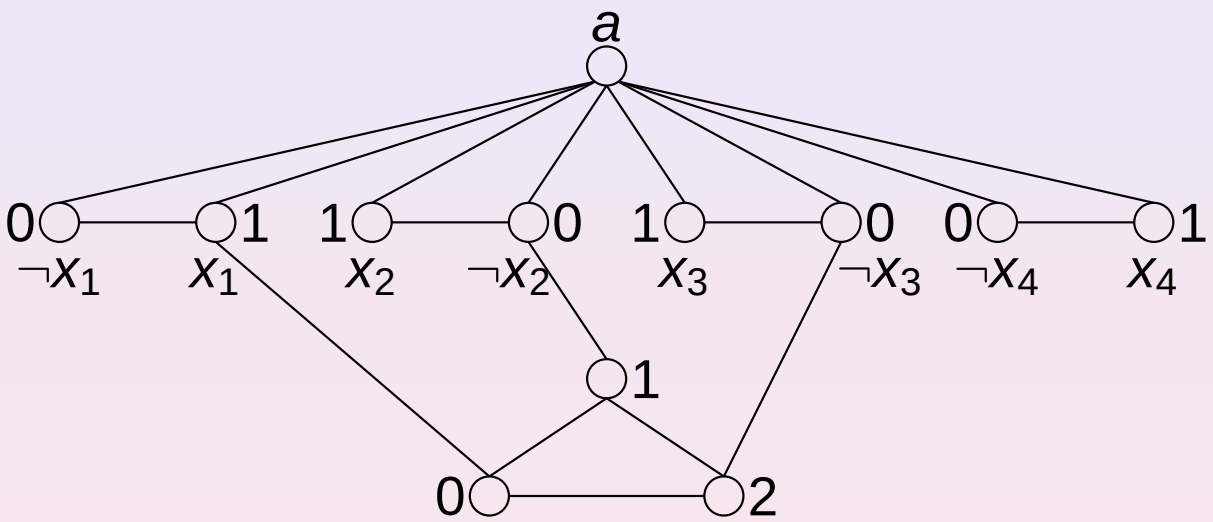
Theorem

3-COLORING *is NP-complete.*

Proof

This is a simple reduction from NAE3SAT. We are given a set of clauses $C_1, \ldots, C_m$, each with three literals, involving the variables $x_1, \ldots, x_n$, and are asked whether there is a truth assignment on the variables such that no clause has all literals true or all literals false. We can construct triangles and use all three colors on the nodes. For each variable x_i, we have a triangle $[a, x_i, \neg x_i]$ where all of the triangles share the node a. Each clause C_i is represented by triangle $[C_{i1}, C_{i2}, C_{i3}]$, where we have an edge connecting C_{ij} with a node that represents the jth literal of C_i. We say that G can be colored with $\{0, 1, 2\}$ if and only if the given instance of NAE3SAT is satisfiable.

3-COLORING (contd.)



Construction for ... $\wedge$ (x_1, $\neg x_2$, $\neg x_3$) $\wedge$ ...

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node *a* have color 2, and for each *i* either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node *a* with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-Coloring (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node *a* have color 2, and for each *i* either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node *a* with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node *a* have color 2, and for each *i* either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node *a* with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-COLORING (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

3-Coloring (contd.)

Proof.

Suppose the graph is 3-colorable. Let node a have color 2, and for each i either x_i or $\neg x_i$ is colored 1 and the other 0. If x_i is 1, it is true, otherwise it is false. If all literals were true, then the corresponding triangle cannot be colored since 1 cannot be used. The same applies when all literals are false. This means there must be at least one literal true and one literal false. Therefore, NAE3SAT is satisfiable. Suppose we have a satisfying truth assignment. We color node a with 2 and the variable triangles in the way that reflects the truth assignment. For any clause, we can pick two literals in it with opposite truth values and color the true literal 1 and the false literal 0. Let the third node be 2. □

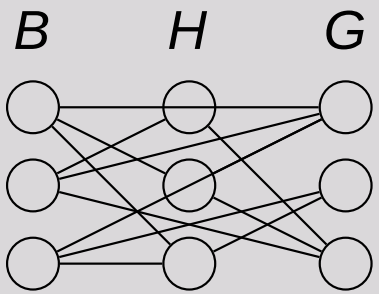
Outline

Tripartite Matching

Problem

Suppose we are given sets B, G, and H (boys, girls, and homes), each containing n elements and a ternary relation $T \subseteq B \times G \times H$. We are asked to find a set of n triples in T, no two of which have a component in common. In other words, each boy is matched to a different girl and each couple has a home of its own.

Example



Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H|}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H|}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)

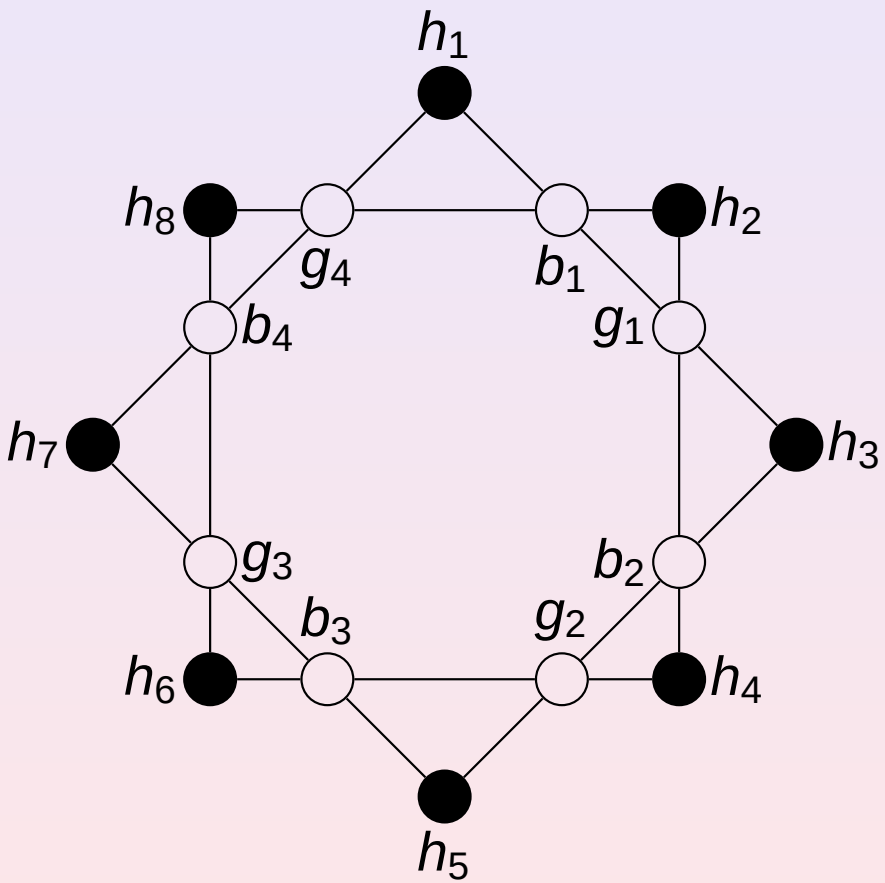
Theorem

Tripartite Matching *is NP-complete.*

Proof.

We reduce 3SAT to it. For each variable x_i, we construct a choice-consistency gadget. Let k be the maximum number of occurrences of x and $\neg x$ (i.e $\max\{occ(x_i), occ(\neg x_i)\}$). We have k boys, k girls, and $2k$ homes in this gadget. For each clause C, we construct a new triple (b, g, h) where h is one of the literals in C, not joined by another triple. Suppose we have m clauses. Since $occ(x_i) + occ(\neg x_i) \leq 2k_i$, we have $3m \leq |H|$, where $|H|$ is the number of homes. This means there are at least $3m$ homes. The number of boys (or girls) is $\frac{|H}{2} + m \leq |H|$. Introduce l more boys and girls such that $|B| = |G| = |H|$. For each of the l boys and girls, add $|H|$ triples that connect to all homes. □

Tripartite Matching (contd.)



The choice-consistency gadget.

Outline

Set Covering

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of finite set U, and a budget B. We want a set of B sets whose union is U.

Corollary

Set Covering *is NP-complete.*

Proof.

This is a generalization of Tripartite Matching. U can be partitioned into three equal sets B, G, and H where each set in F contains one element from each. □

Set Covering

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of finite set U, and a budget B. We want a set of B sets whose union is U.

Corollary

Set Covering *is NP-complete.*

Proof.

This is a generalization of Tripartite Matching. U can be partitioned into three equal sets B, G, and H where each set in F contains one element from each. □

Set Covering

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of finite set U, and a budget B. We want a set of B sets whose union is U.

Corollary

Set Covering *is NP-complete.*

Proof.

This is a generalization of Tripartite Matching. U can be partitioned into three equal sets B, G, and H where each set in F contains one element from each. □

Set Covering

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of finite set U, and a budget B. We want a set of B sets whose union is U.

Corollary

Set Covering *is NP-complete.*

Proof.

This is a generalization of Tripartite Matching. U can be partitioned into three equal sets B, G, and H where each set in F contains one element from each. □

Set Packing

Problem

We are given a family of subsets of a set U and a goal K. We want to see if there are K pairwise disjoint sets in the family.

Corollary

Set Packing *is NP-complete.*

Proof.

This is also a generalization of Tripartite Matching. We partition U into three sets: B, G, and H. We also let $K = 3$ and select one set from each of the partitions. □

Set Packing

Problem

We are given a family of subsets of a set U and a goal K. We want to see if there are K pairwise disjoint sets in the family.

Corollary

Set Packing *is NP-complete.*

Proof.

This is also a generalization of Tripartite Matching. We partition U into three sets: B, G, and H. We also let $K = 3$ and select one set from each of the partitions. □

Set Packing

Problem

We are given a family of subsets of a set U and a goal K. We want to see if there are K pairwise disjoint sets in the family.

Corollary

Set Packing *is NP-complete.*

Proof.

This is also a generalization of Tripartite Matching. We partition U into three sets: B, G, and H. We also let $K = 3$ and select one set from each of the partitions. □

Set Packing

Problem

We are given a family of subsets of a set U and a goal K. We want to see if there are K pairwise disjoint sets in the family.

Corollary

Set Packing *is NP-complete.*

Proof.

This is also a generalization of Tripartite Matching. We partition U into three sets: B, G, and H. We also let $K = 3$ and select one set from each of the partitions. □

Set Packing

Problem

We are given a family of subsets of a set U and a goal K. We want to see if there are K pairwise disjoint sets in the family.

Corollary

Set Packing *is NP-complete.*

Proof.

This is also a generalization of Tripartite Matching. We partition U into three sets: B, G, and H. We also let $K = 3$ and select one set from each of the partitions. □

Exact Cover by 3-Sets

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of a set U where $|U| = 3m$ for some integer m and $|S_i| = 3$ for all i. We want to see if there are m sets in F that are disjoint and have U are their union.

Corollary

Exact Cover by 3-Sets *is NP-complete.*

Proof.

This is a special case of Set Covering where the universe has $3m$ elements, all sets in F have three elements, and the budget is m. □

Exact Cover by 3-Sets

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of a set U where $|U| = 3m$ for some integer m and $|S_i| = 3$ for all i. We want to see if there are m sets in F that are disjoint and have U are their union.

Corollary

Exact Cover by 3-Sets *is NP-complete.*

Proof.

This is a special case of Set Covering where the universe has $3m$ elements, all sets in F have three elements, and the budget is m. □

Exact Cover by 3-Sets

Problem

We are given a family $F = \{S_1, \ldots, S_n\}$ of subsets of a set U where $|U| = 3m$ for some integer m and $|S_i| = 3$ for all i. We want to see if there are m sets in F that are disjoint and have U are their union.

Corollary

Exact Cover by 3-Sets *is NP-complete.*

Proof.

This is a special case of Set Covering where the universe has $3m$ elements, all sets in F have three elements, and the budget is m. □

Outline

Integer Programming

Problem

Given a system of linear inequalities, where we have n variables and integer coefficients, do we have an integer solution?

Theorem

Integer Programming *is NP-Complete*

Proof.

We reduce NAE3SAT to this problem. Suppose we have the clause (x_1, x_2, x_3). We can set up our linear inequalities as $x_1 + x_2 + x_3 \leq 2$, $x_1 + x_2 + x_3 \geq 1$, and $x_i = \{0, 1\}$ where $i = \{1, 2, 3\}$. □

Integer Programming

Problem

Given a system of linear inequalities, where we have n variables and integer coefficients, do we have an integer solution?

Theorem

Integer Programming *is NP-Complete*

Proof.

We reduce NAE3SAT to this problem. Suppose we have the clause (x_1, x_2, x_3). We can set up our linear inequalities as $x_1 + x_2 + x_3 \leq 2$, $x_1 + x_2 + x_3 \geq 1$, and $x_i = \{0, 1\}$ where $i = \{1, 2, 3\}$. □

Integer Programming

Problem

Given a system of linear inequalities, where we have n variables and integer coefficients, do we have an integer solution?

Theorem

Integer Programming *is NP-Complete*

Proof.

We reduce NAE3SAT to this problem. Suppose we have the clause (x_1, x_2, x_3). We can set up our linear inequalities as $x_1 + x_2 + x_3 \leq 2$, $x_1 + x_2 + x_3 \geq 1$, and $x_i = \{0, 1\}$ where $i = \{1, 2, 3\}$. □

Integer Programming

Problem

Given a system of linear inequalities, where we have n variables and integer coefficients, do we have an integer solution?

Theorem

Integer Programming *is NP-Complete*

Proof.

We reduce NAE3SAT to this problem. Suppose we have the clause (x_1, x_2, x_3). We can set up our linear inequalities as $x_1 + x_2 + x_3 \leq 2$, $x_1 + x_2 + x_3 \geq 1$, and $x_i = \{0, 1\}$ where $i = \{1, 2, 3\}$. □

Integer Programming

Problem

Given a system of linear inequalities, where we have n variables and integer coefficients, do we have an integer solution?

Theorem

Integer Programming *is NP-Complete*

Proof.

We reduce NAE3SAT to this problem. Suppose we have the clause (x_1, x_2, x_3). We can set up our linear inequalities as $x_1 + x_2 + x_3 \leq 2$, $x_1 + x_2 + x_3 \geq 1$, and $x_i = \{0, 1\}$ where $i = \{1, 2, 3\}$. □

Outline

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0,1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is 3, 4, 8 and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones. What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones. What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Knapsack Problem

Problem

This is a special case of INTEGER PROGRAMMING. *Select a set of n items, where item i has positive value v_i and positive weight w_i, W is denoted as the weight limit of the items that can be picked, and a goal K. We want to pick the set of items that gives us a total value of K or more without exceeding W.*

Theorem

KNAPSACK *is NP-complete.*

Proof.

We can reduce EXACT COVER BY 3-SETS to it. We are given an instance $\{S_1, S_2, \ldots, S_n\}$ of EXACT COVER BY 3-SETS to see if there are disjoint sets among the sets that cover $U = \{1, 2, \ldots, 3m\}$. Think of the sets as bit vectors in $\{0, 1\}^{3m}$. For example, 001100010 is $3, 4, 8$ and 110010000 is $\{1, 2, 5\}$. Set union would be binary addition. Our goal is to find a subset that gives us all ones.

What if we have a carry? Enlarge the base such that the base is no longer 2, but $n + 1$. We can now get a set of all ones if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Outline

KNAPSACK is Pseudopolynomial

Theorem

Any instance of KNAPSACK *can be solved in* $O(nW)$ *time, where* n *is the number of items and* W *is the weight limit.*

Proof.

We let $V(w, i)$ be the largest value attainable by selecting some of the first i items so that the total weight is exactly w. We can use dynamic programming with each entry of the table as follows: $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

We start by having $V(w, 0) = 0$ for all w. If we encounter $V(W, n) \geq K$, then the given instance of KNAPSACK is "yes". □

KNAPSACK is Pseudopolynomial

Theorem

Any instance of KNAPSACK *can be solved in* $O(nW)$ *time, where* n *is the number of items and* W *is the weight limit.*

Proof.

We let $V(w, i)$ be the largest value attainable by selecting some of the first i items so that the total weight is exactly w. We can use dynamic programming with each entry of the table as follows: $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

We start by having $V(w, 0) = 0$ for all w. If we encounter $V(W, n) \geq K$, then the given instance of KNAPSACK is "yes". □

KNAPSACK is Pseudopolynomial

Theorem

Any instance of KNAPSACK *can be solved in* $O(nW)$ *time, where* n *is the number of items and* W *is the weight limit.*

Proof.

We let $V(w, i)$ be the largest value attainable by selecting some of the first i items so that the total weight is exactly w. We can use dynamic programming with each entry of the table as follows: $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

We start by having $V(w, 0) = 0$ for all w. If we encounter $V(W, n) \geq K$, then the given instance of KNAPSACK is "yes". □

KNAPSACK is Pseudopolynomial

Theorem

Any instance of KNAPSACK *can be solved in* $O(nW)$ *time, where* n *is the number of items and* W *is the weight limit.*

Proof.

We let $V(w, i)$ be the largest value attainable by selecting some of the first i items so that the total weight is exactly w. We can use dynamic programming with each entry of the table as follows: $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

We start by having $V(w, 0) = 0$ for all w. If we encounter $V(W, n) \geq K$, then the given instance of KNAPSACK is "yes". □

KNAPSACK is Pseudopolynomial

Theorem

Any instance of KNAPSACK *can be solved in* $O(nW)$ *time, where* n *is the number of items and* W *is the weight limit.*

Proof.

We let $V(w, i)$ be the largest value attainable by selecting some of the first i items so that the total weight is exactly w. We can use dynamic programming with each entry of the table as follows: $V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$

We start by having $V(w, 0) = 0$ for all w. If we encounter $V(W, n) \geq K$, then the given instance of KNAPSACK is "yes". □

KNAPSACK is Pseudopolynomial (contd.)

Observation

Note that KNAPSACK *is not polynomial because the time bound* nW *is not polynomial. The length of the input is more like* $n \log W$.

Definition

A problem is *strongly* NP-complete if the problem remains NP-complete if any instance of length n is restricted to contain integers of size at most $p(n)$, a polynomial.

KNAPSACK is Pseudopolynomial (contd.)

Observation

Note that KNAPSACK *is not polynomial because the time bound* nW *is not polynomial. The length of the input is more like* $n \log W$.

Definition

A problem is *strongly* NP-complete if the problem remains NP-complete if any instance of length n is restricted to contain integers of size at most $p(n)$, a polynomial.

Subset Sum

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

Subset Sum *is NP-complete.*

Proof.

We reduce Knapsack to this problem. Given an instance of Knapsack, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for Subset Sum. □

Subset Sum

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

Subset Sum *is NP-complete.*

Proof.

We reduce Knapsack to this problem. Given an instance of Knapsack, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for Subset Sum. □

SUBSET SUM

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

SUBSET SUM *is NP-complete.*

Proof.

We reduce KNAPSACK to this problem. Given an instance of KNAPSACK, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for SUBSET SUM. □

Subset Sum

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

Subset Sum *is NP-complete.*

Proof.

We reduce Knapsack to this problem. Given an instance of Knapsack, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for Subset Sum. □

Subset Sum

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

Subset Sum *is NP-complete.*

Proof.

We reduce Knapsack to this problem. Given an instance of Knapsack, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for Subset Sum. □

SUBSET SUM

Problem

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ and value T, is there a subset $S' \subseteq S$ such that $\sum_{a_i \in S'} a_i = T$?

Lemma

SUBSET SUM *is NP-complete.*

Proof.

We reduce KNAPSACK to this problem. Given an instance of KNAPSACK, we have n items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, a weight limit W and a goal K. We want to solve the equations $\sum w_i x_i \leq W$ and $\sum v_i x_i \geq K$ where $x_i = \{0, 1\}$. We simply let $w_i = v_i$ and $W = K$ to give us the equation $\sum w_i x_i = K$. A solution to this instance exists if and only if a solution exists for SUBSET SUM. □

Bin Packing

Problem

We are given N positive integers $a_1, a_2, \ldots, a_N$ (the items), and two more integers C (the capacity) and B (the number of bins). We want to see if we can partition the numbers into B subsets, each of which has a total sum at most C.

Theorem

Bin Packing *is NP-complete.*

Proof

We reduce Subset Sum to this problem. Given an instance of Subset Sum with the set $S = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \leq a_2 \leq \ldots \leq a_n$, we let $S_1 = \{a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}\}$, $S_2 = \{a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n\}$, and $T = \sum_{a_i \in S_2} a_i$. We let $B = 2$ and $C = T$. This gives us a solution to the Bin Packing problem if and only if we have a solution for Subset Sum.

Bin Packing

Problem

We are given N positive integers $a_1, a_2, \ldots, a_N$ (the items), and two more integers C (the capacity) and B (the number of bins). We want to see if we can partition the numbers into B subsets, each of which has a total sum at most C.

Theorem

Bin Packing *is NP-complete.*

Proof

We reduce Subset Sum to this problem. Given an instance of Subset Sum with the set $S = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \leq a_2 \leq \ldots \leq a_n$, we let $S_1 = \{a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}\}$, $S_2 = \{a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n\}$, and $T = \sum_{a_i \in S_2} a_i$. We let $B = 2$ and $C = T$. This gives us a solution to the Bin Packing problem if and only if we have a solution for Subset Sum.

Bin Packing

Problem

We are given N positive integers $a_1, a_2, \ldots, a_N$ (the items), and two more integers C (the capacity) and B (the number of bins). We want to see if we can partition the numbers into B subsets, each of which has a total sum at most C.

Theorem

Bin Packing *is NP-complete.*

Proof

We reduce Subset Sum to this problem. Given an instance of Subset Sum with the set $S = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \leq a_2 \leq \ldots \leq a_n$, we let $S_1 = \{a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}\}$, $S_2 = \{a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n\}$, and $T = \sum_{a_i \in S_2} a_i$. We let $B = 2$ and $C = T$. This gives us a solution to the Bin Packing problem if and only if we have a solution for Subset Sum.

Bin Packing

Problem

We are given N positive integers $a_1, a_2, \ldots, a_N$ (the items), and two more integers C (the capacity) and B (the number of bins). We want to see if we can partition the numbers into B subsets, each of which has a total sum at most C.

Theorem

Bin Packing *is NP-complete.*

Proof

We reduce Subset Sum to this problem. Given an instance of Subset Sum with the set $S = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \leq a_2 \leq \ldots \leq a_n$, we let $S_1 = \{a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}\}$, $S_2 = \{a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n\}$, and $T = \sum_{a_i \in S_2} a_i$. We let $B = 2$ and $C = T$. This gives us a solution to the Bin Packing problem if and only if we have a solution for Subset Sum.

Bin Packing

Problem

We are given N positive integers $a_1, a_2, \ldots, a_N$ (the items), and two more integers C (the capacity) and B (the number of bins). We want to see if we can partition the numbers into B subsets, each of which has a total sum at most C.

Theorem

Bin Packing *is NP-complete.*

Proof

We reduce Subset Sum to this problem. Given an instance of Subset Sum with the set $S = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \leq a_2 \leq \ldots \leq a_n$, we let $S_1 = \{a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}\}$, $S_2 = \{a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n\}$, and $T = \sum_{a_i \in S_2} a_i$. We let $B = 2$ and $C = T$. This gives us a solution to the Bin Packing problem if and only if we have a solution for Subset Sum.

NP-Complete Reduction Hierarchy