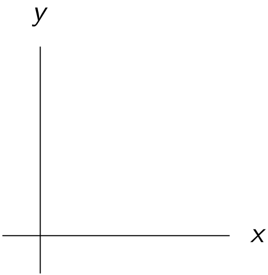
How the simplex algorithm works
COMS20010 2020, Video 8-2

John Lapinskas, University of Bristol

How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$x + y \to \max$ subject to

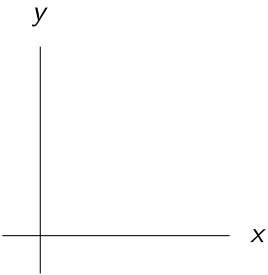


How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$x + y \to \max$ subject to

$x + 2y \leq 4$;

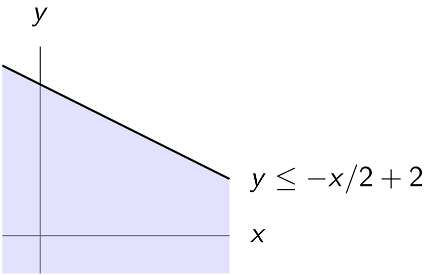


How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$x + y \to \max$ subject to

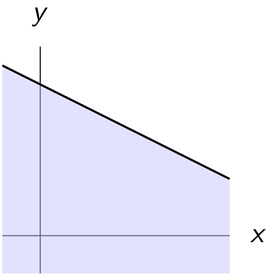
$x + 2y \leq 4$;



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

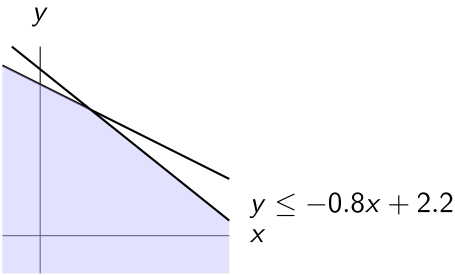
$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

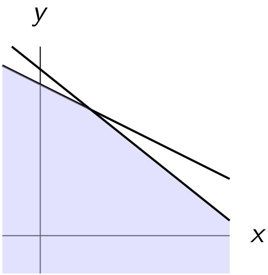
$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

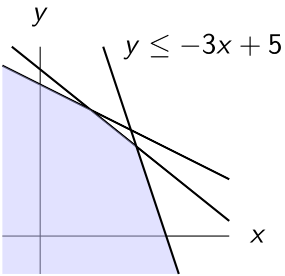
$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \\ 3x + y &\leq 5; \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

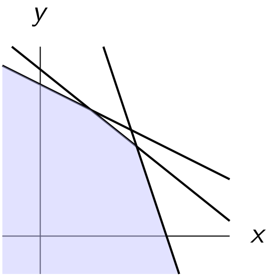
$$x + y \to \max \text{ subject to}$$
$$x + 2y \leq 4;$$
$$4x + 5y \leq 11;$$
$$3x + y \leq 5;$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

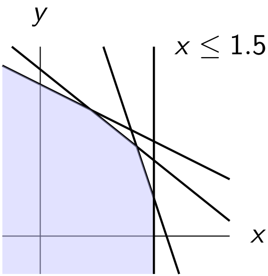
$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \\ 3x + y &\leq 5; \\ 2x &\leq 3; \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

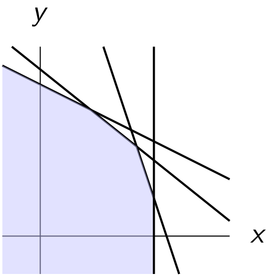
$$\begin{aligned} x + y \to \max\ &\text{subject to} \\ x + 2y &\le 4; \\ 4x + 5y &\le 11; \\ 3x + y &\le 5; \\ 2x &\le 3; \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

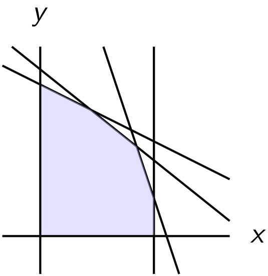
$$\begin{aligned} x + y &\to \max \ \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \\ 3x + y &\leq 5; \\ 2x &\leq 3; \\ x,\ y &\geq 0. \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

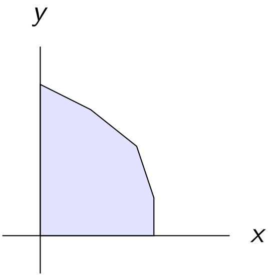
$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\le 4; \\ 4x + 5y &\le 11; \\ 3x + y &\le 5; \\ 2x &\le 3; \\ x,\ y &\ge 0. \end{aligned}$$



How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\le 4; \\ 4x + 5y &\le 11; \\ 3x + y &\le 5; \\ 2x &\le 3; \\ x,\ y &\ge 0. \end{aligned}$$

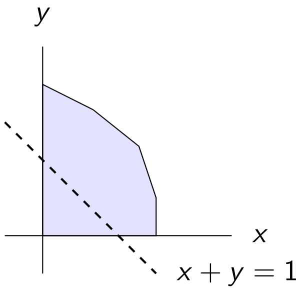


Let f be the objective function. Then for all c, $\{\vec{x} \in \mathbb{R}^n \colon f(\vec{x}) = c\}$ is an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$. The problem reduces to: how large can we take c and still have this hyperplane intersect the feasible polytope?

How to solve linear programs?

We can look at linear programs geometrically. The *n*-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$$
\begin{aligned}
x + y &\to \max \ \text{ subject to} \\
x + 2y &\le 4; \\
4x + 5y &\le 11; \\
3x + y &\le 5; \\
2x &\le 3; \\
x,\ y &\ge 0.
\end{aligned}
$$



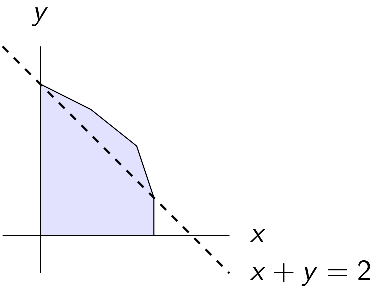
Let f be the objective function. Then for all c, $\{\vec{x} \in \mathbb{R}^n \colon f(\vec{x}) = c\}$ is an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$. The problem reduces to: how large can we take c and still have this hyperplane intersect the feasible polytope?

“Corollary”: There will always be an optimal solution at a corner!

How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$$
\begin{aligned}
x + y &\to \max \ \text{ subject to} \\
x + 2y &\leq 4; \\
4x + 5y &\leq 11; \\
3x + y &\leq 5; \\
2x &\leq 3; \\
x,\ y &\geq 0.
\end{aligned}
$$



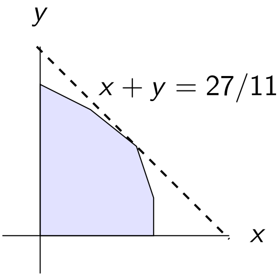
Let f be the objective function. Then for all c, $\{\vec{x} \in \mathbb{R}^n \colon f(\vec{x}) = c\}$ is an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$. The problem reduces to: how large can we take c and still have this hyperplane intersect the feasible polytope?

"Corollary": There will always be an optimal solution at a corner!

How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$$\begin{aligned} x + y \to \max \ &\text{subject to} \\ x + 2y &\le 4; \\ 4x + 5y &\le 11; \\ 3x + y &\le 5; \\ 2x &\le 3; \\ x,\ y &\ge 0. \end{aligned}$$



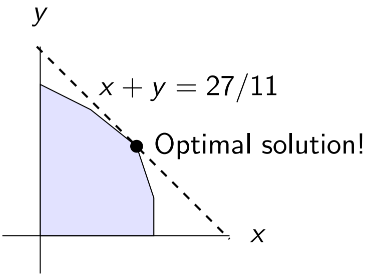
Let f be the objective function. Then for all c, $\{\vec{x} \in \mathbb{R}^n : f(\vec{x}) = c\}$ is an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$. The problem reduces to: how large can we take c and still have this hyperplane intersect the feasible polytope?

"Corollary": There will always be an optimal solution at a corner!

How to solve linear programs?

We can look at linear programs geometrically. The n-variable constraints describe a feasible polytope in $\mathbb{R}^n$. For example, if $n = 2$:

$$\begin{aligned} x + y &\to \max \text{ subject to} \\ x + 2y &\leq 4; \\ 4x + 5y &\leq 11; \\ 3x + y &\leq 5; \\ 2x &\leq 3; \\ x, y &\geq 0. \end{aligned}$$



Let f be the objective function. Then for all c, $\{\vec{x} \in \mathbb{R}^n : f(\vec{x}) = c\}$ is an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$. The problem reduces to: how large can we take c and still have this hyperplane intersect the feasible polytope?

"Corollary": There will always be an optimal solution at a corner!

The simplex algorithm

The n-variable constraints of an LP describe a feasible polytope in $\mathbb{R}^n$.

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm

The n-variable constraints of an LP describe a feasible polytope in $\mathbb{R}^n$.

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm can be described in great and tedious detail, but the point is: search **greedily** for a vertex of the feasible polytope which maximises the objective function.

The simplex algorithm

The n-variable constraints of an LP describe a feasible polytope in $\mathbb{R}^n$.

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm can be described in great and tedious detail, but the point is: search **greedily** for a vertex of the feasible polytope which maximises the objective function.

So starting at an arbitrary vertex, look at all the neighbouring vertices and move to whichever one makes the objective function biggest.

If none of them do, return the current vertex.

The simplex algorithm

The n-variable constraints of an LP describe a feasible polytope in $\mathbb{R}^n$.

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm can be described in great and tedious detail, but the point is: search **greedily** for a vertex of the feasible polytope which maximises the objective function.

So starting at an arbitrary vertex, look at all the neighbouring vertices and move to whichever one makes the objective function biggest.

If none of them do, return the current vertex.

Problem: There are often $\Omega(2^n)$ vertices, e.g. with a hypercube!

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

Intuitively, it feels like there should always be a short path from any vertex to an optimal solution...

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

Intuitively, it feels like there should always be a short path from any vertex to an optimal solution... but the simplex algorithm need not find it. Klee and Minty (1973) gave a "twisted hypercube" on which the simplex algorithm needed to visit **every single vertex**.

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

Intuitively, it feels like there should always be a short path from any vertex to an optimal solution... but the simplex algorithm need not find it. Klee and Minty (1973) gave a “twisted hypercube” on which the simplex algorithm needed to visit **every single vertex**.

So why bother with it if it might take exponentially long?

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

Intuitively, it feels like there should always be a short path from any vertex to an optimal solution... but the simplex algorithm need not find it. Klee and Minty (1973) gave a "twisted hypercube" on which the simplex algorithm needed to visit **every single vertex**.

So why bother with it if it might take exponentially long?

Because in practice it normally only needs $\Theta(n)$ steps! Explaining why this is true is a major open problem in theoretical computer science.

Running time of the simplex algorithm

If the linear program *has* an optimal solution, i.e. if it is bounded and the feasible polytope is non-empty, then it will have one at a **vertex** (i.e. corner) of the polytope.

The simplex algorithm works by searching through the vertices greedily.

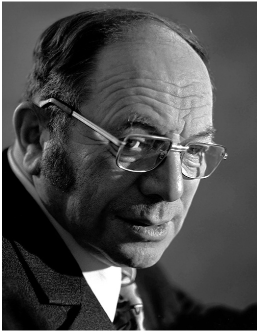
Intuitively, it feels like there should always be a short path from any vertex to an optimal solution... but the simplex algorithm need not find it. Klee and Minty (1973) gave a “twisted hypercube” on which the simplex algorithm needed to visit **every single vertex**.

So why bother with it if it might take exponentially long?

Because in practice it normally only needs $\Theta(n)$ steps! Explaining why this is true is a major open problem in theoretical computer science.

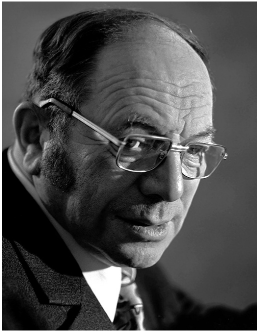
There are also **interior point algorithms**, which have a polynomial worst-case run-time, but which generally work less well in practice.

How not to do it...



Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

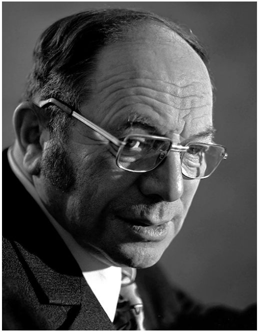
How not to do it...



Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

This later became the basis of the entire Soviet planned economy, for which he received the Stalin Award in 1949.

How not to do it...

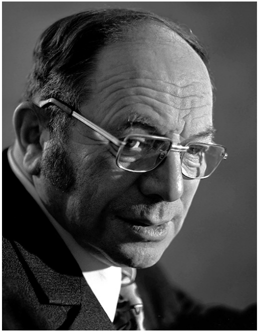


Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

This later became the basis of the entire Soviet planned economy, for which he received the Stalin Award in 1949.

The Soviet planned economy didn't do so well! So what went wrong?

How not to do it...



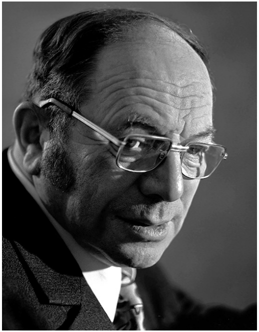
Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

This later became the basis of the entire Soviet planned economy, for which he received the Stalin Award in 1949.

The Soviet planned economy didn't do so well! So what went wrong?

- Lack of computational power led to awful simplifying assumptions.

How not to do it...



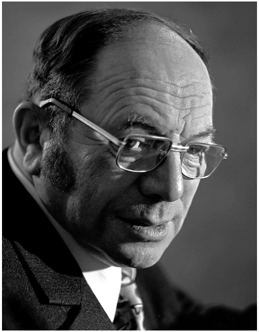
Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

This later became the basis of the entire Soviet planned economy, for which he received the Stalin Award in 1949.

The Soviet planned economy didn't do so well! So what went wrong?

- Lack of computational power led to awful simplifying assumptions.
- They had no accurate data.

How not to do it...



Linear programming was first proposed by Leonid Kantorovich to solve the problem of most effectively producing as much plywood as possible in a specific trust.

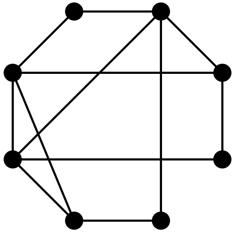
This later became the basis of the entire Soviet planned economy, for which he received the Stalin Award in 1949.

The Soviet planned economy didn't do so well! So what went wrong?

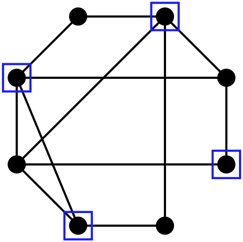
- Lack of computational power led to awful simplifying assumptions.
- They had no accurate data.
- What should the objective function be?

A less lethal application: Approximation algorithms

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

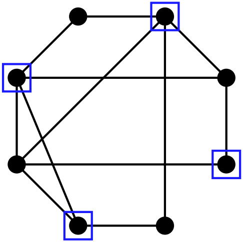


A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.



A less lethal application: Approximation algorithms

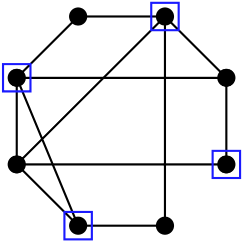
A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.



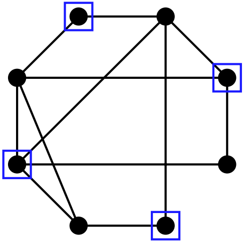
A valid vertex cover.

A less lethal application: Approximation algorithms

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

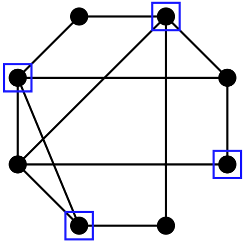


A valid vertex cover.

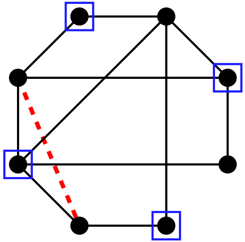


A less lethal application: Approximation algorithms

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

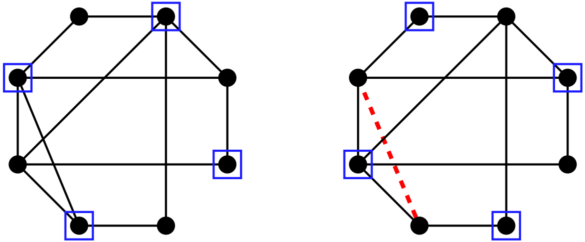


A valid vertex cover.



Not a valid vertex cover.

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.



A valid vertex cover. **Not** a valid vertex cover.

We would like to find the **smallest possible** vertex cover of G.

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an **integer linear program**.

Given a graph $G = (V, E)$, we assign a variable $x_v \in \{0, 1\}$ to each vertex v. We interpret $x_v = 1$ as "v is in the cover", and $x_v = 0$ as "v is not in the cover". We can then formulate the problem as:

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an **integer linear program**.

Given a graph $G = (V, E)$, we assign a variable $x_v \in \{0, 1\}$ to each vertex v. We interpret $x_v = 1$ as "v is in the cover", and $x_v = 0$ as "v is not in the cover". We can then formulate the problem as:

$\sum_i x_i \to \min$ subject to

Minimise $|X|$ subject to

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an **integer linear program**.

Given a graph $G = (V, E)$, we assign a variable $x_v \in \{0, 1\}$ to each vertex v. We interpret $x_v = 1$ as "v is in the cover", and $x_v = 0$ as "v is not in the cover". We can then formulate the problem as:

$$\begin{aligned} &\textstyle\sum_i x_i \to \min \text{ subject to} \\ &x_u + x_v \geq 1 \text{ for all } \{u, v\} \in E; \end{aligned}$$

$$\begin{aligned} &\text{Minimise } |X| \text{ subject to} \\ &u \in X \text{ or } v \in X \text{ (or both)} \\ &\text{for all } \{u, v\} \in E \end{aligned}$$

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an **integer linear program**.

Given a graph $G = (V, E)$, we assign a variable $x_v \in \{0, 1\}$ to each vertex v. We interpret $x_v = 1$ as "v is in the cover", and $x_v = 0$ as "v is not in the cover". We can then formulate the problem as:

$\sum_i x_i \to \min$ subject to	Minimise $\lvert X\rvert$ subject to
$x_u + x_v \geq 1$ for all $\{u, v\} \in E$;	$u \in X$ or $v \in X$ (or both) for all $\{u, v\} \in E$
$x_v \leq 1$ for all $v \in V$;	
$x_v \geq 0$ for all $v \in V$;	[Ensures $x_v \in \{0, 1\}$ for all v]
$x_v \in \mathbb{N}$ for all $v \in V$.	

A **vertex cover** in a graph $G = (V, E)$ is a set $X \subseteq V$ such that every edge in E has at least one vertex in X.

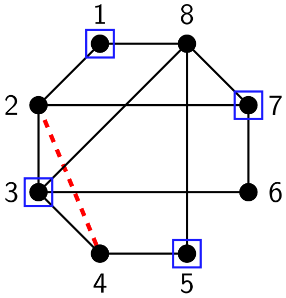
We can express finding a minimum vertex cover as solving a linear program in which the solutions must be integers: an **integer linear program**.

Given a graph $G = (V, E)$, we assign a variable $x_v \in \{0, 1\}$ to each vertex v. We interpret $x_v = 1$ as "v is in the cover", and $x_v = 0$ as "v is not in the cover". We can then formulate the problem as:

$\sum_i x_i \to \min$ subject to	Minimise $\lvert X\rvert$ subject to
$x_u + x_v \geq 1$ for all $\{u, v\} \in E$;	$u \in X$ or $v \in X$ (or both) for all $\{u, v\} \in E$
$x_v \leq 1$ for all $v \in V$;	
$x_v \geq 0$ for all $v \in V$;	[Ensures $x_v \in \{0, 1\}$ for all v]
$x_v \in \mathbb{N}$ for all $v \in V$.	

Optimal solutions of this ILP correspond to minimum vertex covers of G, and minimum vertex covers of G correspond to optimal solutions.

An example of the ILP formulation of vertex cover



$$\textstyle\sum_v x_v \to \min \text{ subject to}$$
$$x_u + x_v \geq 1 \text{ for all } \{u, v\} \in E;$$
$$x_v \leq 1 \text{ for all } v \in V;$$
$$x_v \geq 0 \text{ for all } v \in V;$$
$$x_v \in \mathbb{N} \text{ for all } v \in V.$$

$X = \{1, 3, 5, 7\}$ is **not** a vertex cover.

Here we have $x_1 = x_3 = x_5 = x_7 = 1$ and $x_0 = x_2 = x_4 = x_6 = 0$.

The uncovered edge $\{2, 4\}$ corresponds to the constraint $x_2 + x_4 \geq 1$, which is violated.

So now we just solve the ILP, right?

So now we just solve the ILP, right?

No. As we'll see later, it's generally impossible to solve ILPs efficiently. It's also impossible to find a minimum vertex cover efficiently. :-(

So now we just solve the ILP, right?

No. As we'll see later, it's generally impossible to solve ILPs efficiently. It's also impossible to find a minimum vertex cover efficiently. :-(

But we *can* solve LPs. So what if we **relax** our ILP by allowing non-integer solutions, turning it into an LP, and solve that?

So now we just solve the ILP, right?

No. As we'll see later, it's generally impossible to solve ILPs efficiently. It's also impossible to find a minimum vertex cover efficiently. :-(

But we *can* solve LPs. So what if we **relax** our ILP by allowing non-integer solutions, turning it into an LP, and solve that?

$$\begin{aligned} \textstyle\sum_i x_i &\to \min \text{ subject to} \\ x_u + x_v &\geq 1 \text{ for all } \{u, v\} \in E; \\ x_v &\leq 1 \text{ for all } v \in V; \\ x_v &\geq 0 \text{ for all } v \in V. \end{aligned}$$

Then we take our vertex cover X to be $\{v \in V : x_v \geq 1/2\}$, essentially rounding up to recover a feasible solution for the ILP.

So now we just solve the ILP, right?

No. As we'll see later, it's generally impossible to solve ILPs efficiently. It's also impossible to find a minimum vertex cover efficiently. :-(

But we *can* solve LPs. So what if we **relax** our ILP by allowing non-integer solutions, turning it into an LP, and solve that?

$$\begin{aligned}\textstyle\sum_i x_i &\to \min \text{ subject to}\\ x_u + x_v &\geq 1 \text{ for all } \{u, v\} \in E;\\ x_v &\leq 1 \text{ for all } v \in V;\\ x_v &\geq 0 \text{ for all } v \in V.\end{aligned}$$

Then we take our vertex cover X to be $\{v \in V : x_v \geq 1/2\}$, essentially rounding up to recover a feasible solution for the ILP.

It's not hard to show (see problem sheet) that if a minimum vertex cover has size k, then X is indeed a vertex cover and $k \leq |X| \leq 2k$. So even though the problem is hard, we can still find an **approximate** solution!