Computational Geometry

Linear Programs & Half-Plane Intersection

Thomas Bläsius

Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies

Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused

Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused

For Which Objects Is There a Permanent Mold?

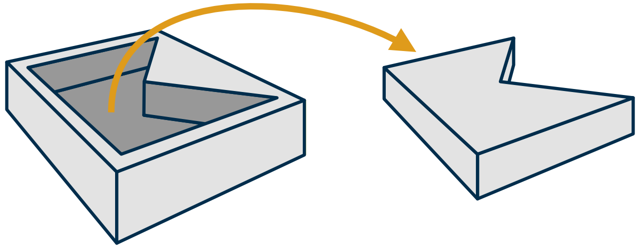
Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused

For Which Objects Is There a Permanent Mold?

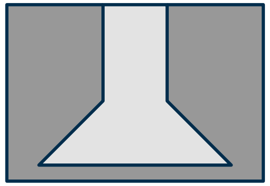
- assumption: mold consists of one piece



Developing A Mold

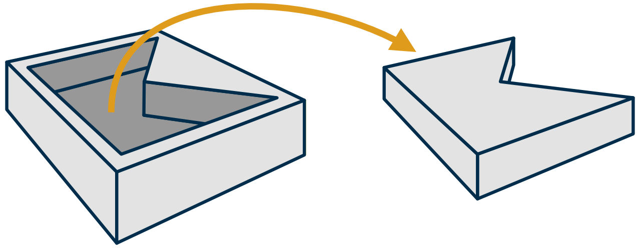
Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused



For Which Objects Is There a Permanent Mold?

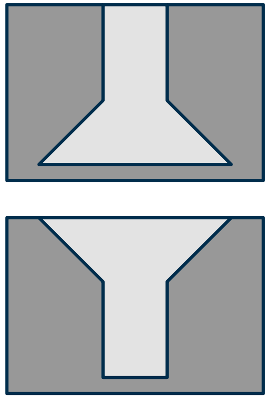
- assumption: mold consists of one piece
- the object may be stuck in the mold

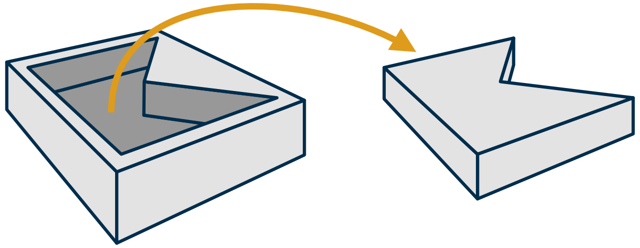


Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused





For Which Objects Is There a Permanent Mold?

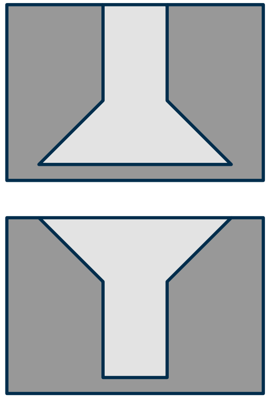
- assumption: mold consists of one piece
- the object may be stuck in the mold
- but a different mold for the same object works

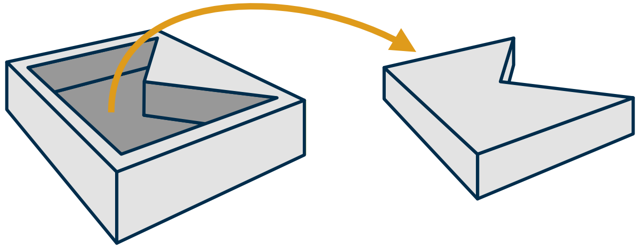
Developing A Mold

Casting (Manufacturing Technique)

- liquid material is poured into a mold, where it solidifies
- expendable mold: gets destroyed when removing the object
- permanent mold: remains intact and can be reused

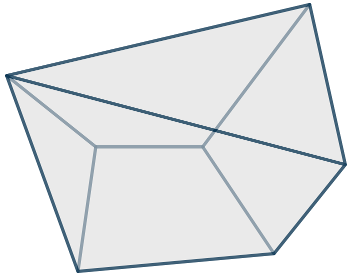
For Which Objects Is There a Permanent Mold?





- assumption: mold consists of one piece
- the object may be stuck in the mold
- but a different mold for the same object works

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.



Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

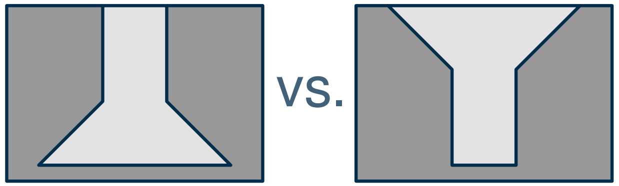
Decisions To Make

Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

- choice of the top face

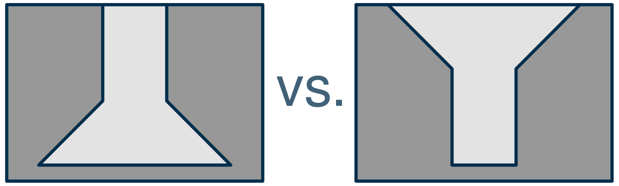


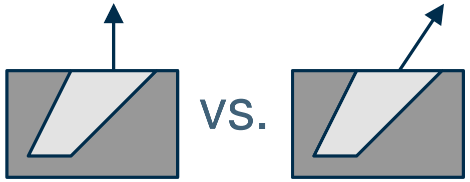
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

- choice of the top face
- direction of the translation



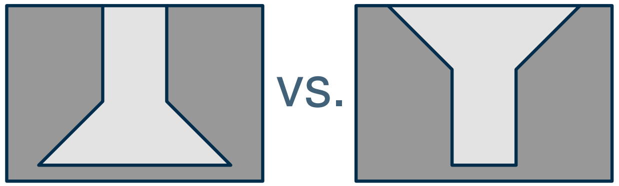


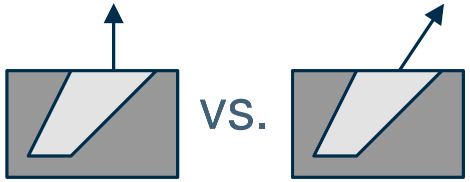
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

- choice of the top face
- direction of the translation





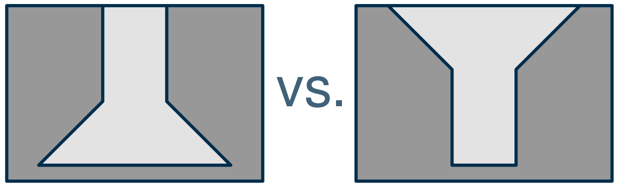
Simplified Situation: Top Face Already Selected

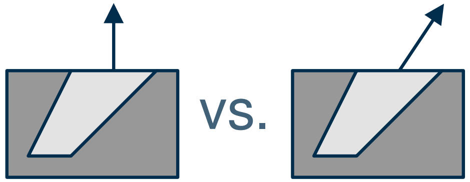
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

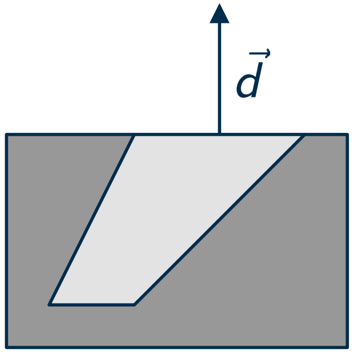
- choice of the top face
- direction of the translation





Simplified Situation: Top Face Already Selected

- direction of the translation: $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$

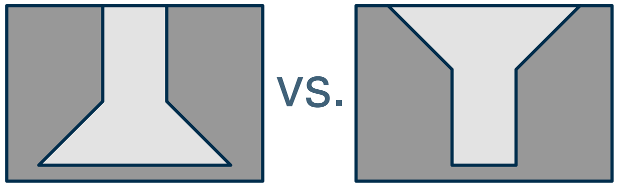


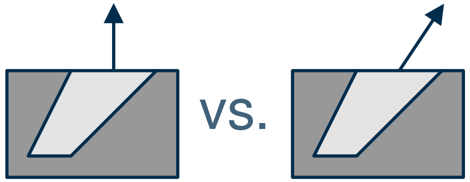
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

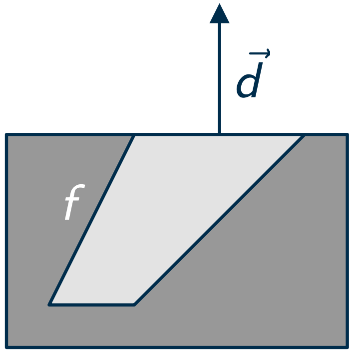
- choice of the top face
- direction of the translation





Simplified Situation: Top Face Already Selected

- direction of the translation: $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- let f be a regular face f (not the top face)

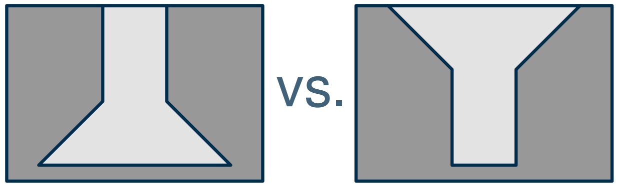


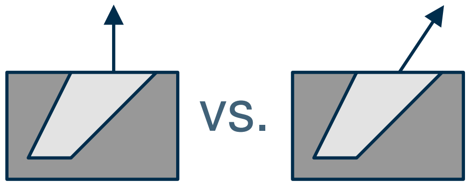
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

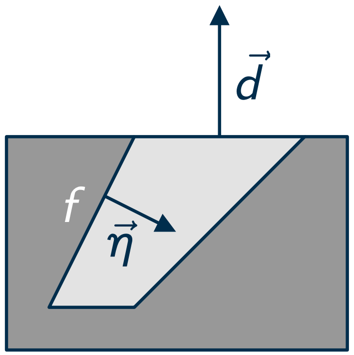
- choice of the top face
- direction of the translation





Simplified Situation: Top Face Already Selected

- direction of the translation: $\vec{d} = \begin{pmatrix} d_x & d_y & d_z \end{pmatrix}^T \in \mathbb{R}^3$
- let f be a regular face f (not the top face)
- let $\vec{\eta}$ be its normal vector (pointing inwards)

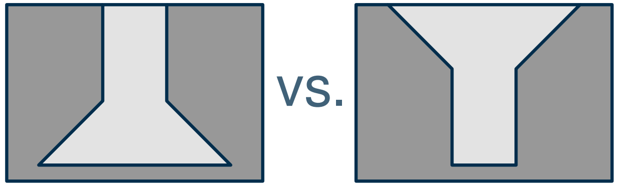


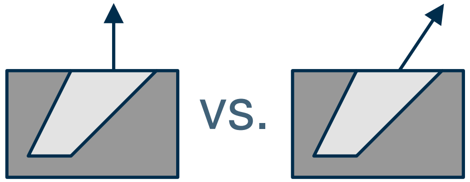
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

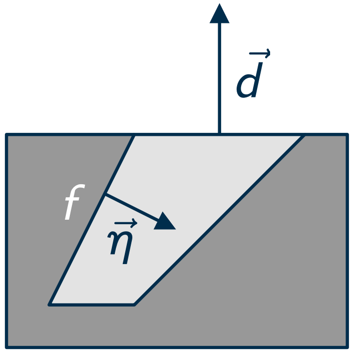
- choice of the top face
- direction of the translation





Simplified Situation: Top Face Already Selected

- direction of the translation: $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- let f be a regular face f (not the top face)
- let $\vec{\eta}$ be its normal vector (pointing inwards)
- problem in the example: angle between $\vec{d}$ and $\vec{\eta}$ is bigger than 90°

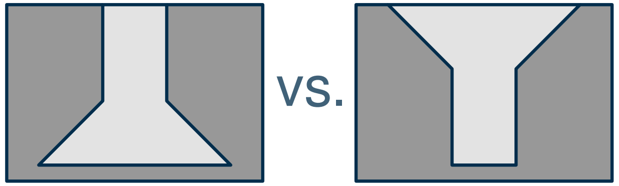


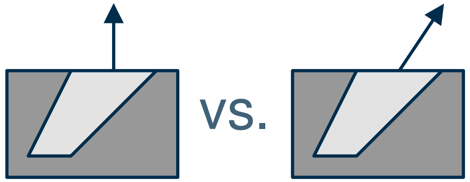
Initial Observations

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Decisions To Make

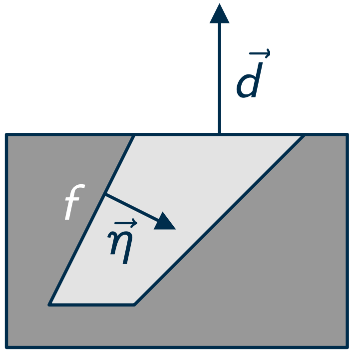
- choice of the top face
- direction of the translation





Simplified Situation: Top Face Already Selected

- direction of the translation: $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- let f be a regular face f (not the top face)
- let $\vec{\eta}$ be its normal vector (pointing inwards)
- problem in the example: angle between $\vec{d}$ and $\vec{\eta}$ is bigger than 90°



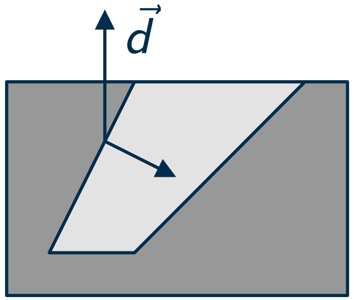
Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90^\circ$ for every regular face.

Good And Bad Translations

Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90°$ for every regular face.

Proof

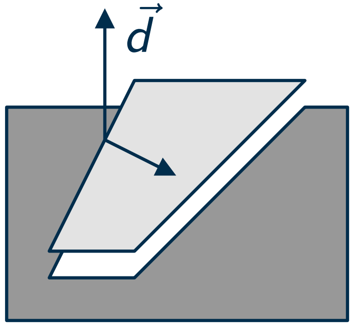
- angle $> 90° \Rightarrow$ cannot be removed
- angle $\leq 90° \Rightarrow$ can be removed



Good And Bad Translations

Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90^\circ$ for every regular face.

Proof



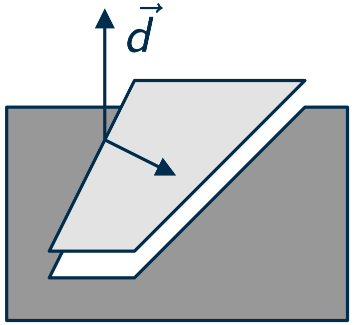
- angle $> 90^\circ \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90^\circ \Rightarrow$ can be removed

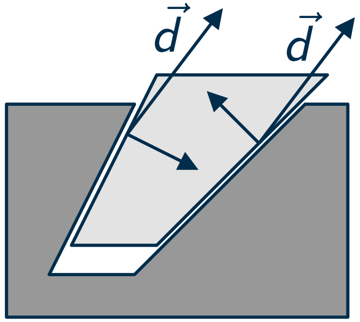
Good And Bad Translations

Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90^\circ$ for every regular face.

Proof

- angle $> 90^\circ \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90^\circ \Rightarrow$ can be removed
  - ok at the beginning



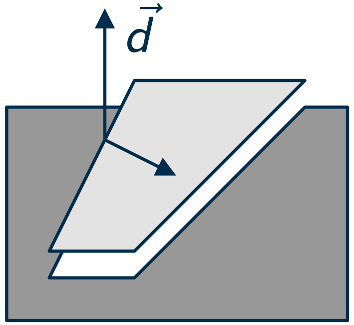


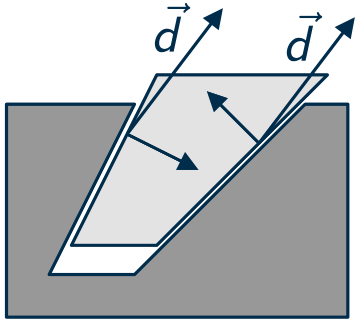
Good And Bad Translations

Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90^\circ$ for every regular face.

Proof

- angle $> 90^\circ \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90^\circ \Rightarrow$ can be removed
  - ok at the beginning
  - can the polyhedron collide with the mold later?



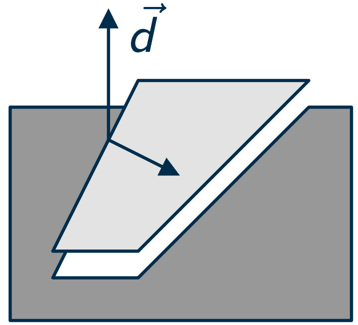


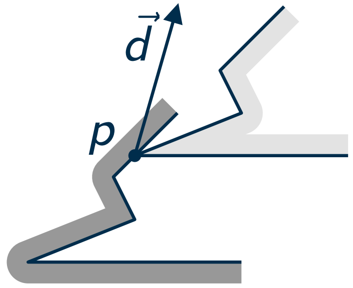
Good And Bad Translations

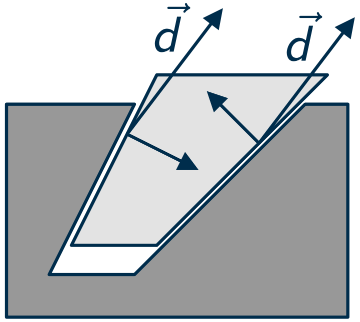
Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90°$ for every regular face.

Proof

- angle $> 90° \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90° \Rightarrow$ can be removed
  - ok at the beginning
  - can the polyhedron collide with the mold later?
  - let p be the first point of P that collides with the mold





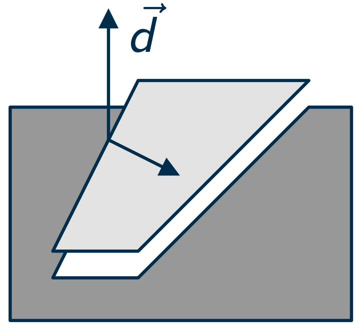


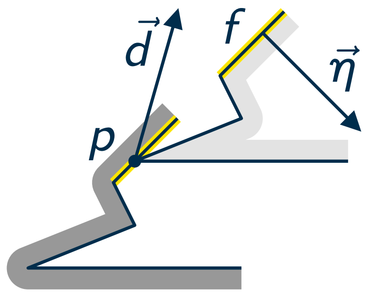
Good And Bad Translations

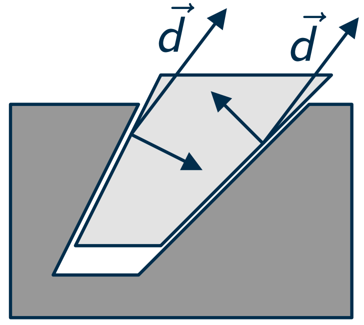
Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90°$ for every regular face.

Proof

- angle $> 90° \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90° \Rightarrow$ can be removed
  - ok at the beginning
  - can the polyhedron collide with the mold later?
  - let p be the first point of P that collides with the mold
  - let f be the corresponding face of P with normal $\vec{\eta}$



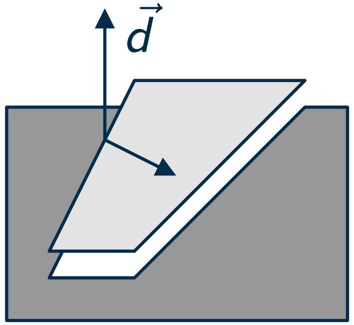


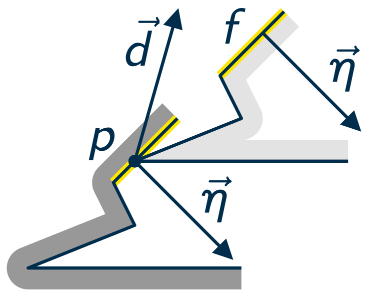


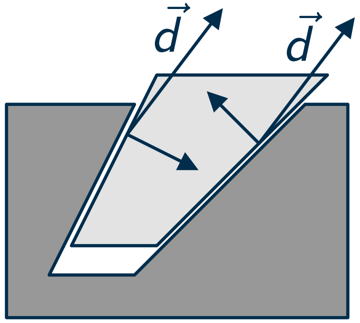
Good And Bad Translations

Lemma
P can be removed in the direction $\vec{d}$ if and only if the angle between $\vec{d}$ and the inner normal is $\leq 90^\circ$ for every regular face.

Proof
- angle $> 90^\circ \Rightarrow$ cannot be removed
  - already fails right at the beginning of the translation
- angle $\leq 90^\circ \Rightarrow$ can be removed
  - ok at the beginning
  - can the polyhedron collide with the mold later?
  - let p be the first point of P that collides with the mold
  - let f be the corresponding face of P with normal $\vec{\eta}$
  - angle between $\vec{d}$ and $\vec{\eta}$: $> 90^\circ$







What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

Reminder (Dot Product): angle between $\vec{d}$ and $\vec{\eta} \leq 90^\circ \Leftrightarrow \vec{d} \cdot \vec{\eta} \geq 0$

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = \begin{pmatrix} d_x & d_y & d_z \end{pmatrix}^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

Reminder (Dot Product): angle between $\vec{d}$ and $\vec{\eta} \leq 90^\circ \Leftrightarrow \vec{d} \cdot \vec{\eta} \geq 0$

Restating The Problem (For A Fixed Upper Face)

- find d_x and d_y

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = \begin{pmatrix} d_x & d_y & d_z \end{pmatrix}^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

Reminder (Dot Product): angle between $\vec{d}$ and $\vec{\eta} \leq 90^\circ \Leftrightarrow \vec{d} \cdot \vec{\eta} \geq 0$

Restating The Problem (For A Fixed Upper Face)

- find d_x and d_y
- such that for every regular face we have: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

Reminder (Dot Product): angle between $\vec{d}$ and $\vec{\eta} \leq 90^\circ \Leftrightarrow \vec{d} \cdot \vec{\eta} \geq 0$

Restating The Problem (For A Fixed Upper Face)

- find d_x and d_y
- such that for every regular face we have: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- this is a linear program (LP)

What Do We Need To Do?

Problem
Given a polyhedron P, is there a mold for P from which P can be removed with a translation.

Goal

- choose upper face for P
- choose a direction $\vec{d} = (d_x \quad d_y \quad d_z)^T \in \mathbb{R}^3$
- such that for every normal $\vec{\eta}$ of a regular face: angle between $\vec{d}$ and $\vec{\eta}$ is at most 90°
- note: we can assume $d_z = 1$

Why?

Reminder (Dot Product): angle between $\vec{d}$ and $\vec{\eta} \leq 90^\circ \Leftrightarrow \vec{d} \cdot \vec{\eta} \geq 0$

Restating The Problem (For A Fixed Upper Face)

- find d_x and d_y
- such that for every regular face we have: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- this is a linear program (LP)

Is the inequality really linear?

Linear Programs

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- an objective function
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$
\begin{aligned}
\text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\
\text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\
& a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\
& \qquad \vdots \\
& a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n
\end{aligned}
$$

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- an objective function
- d is the dimension of the LP
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$
\begin{aligned}
\text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\
\text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\
& a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\
& \vdots \\
& a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n
\end{aligned}
$$

Linear Programs

General Form Of An LP

- variables $x_1, \dots, x_d$
- an objective function
- d is the dimension of the LP
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$\begin{aligned} \text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\ \text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\ & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\ & \vdots \\ & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n \end{aligned}$$

Our Specific LP

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- an objective function
- d is the dimension of the LP
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$
\begin{aligned}
\text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\
\text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\
& a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\
& \vdots \\
& a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n
\end{aligned}
$$

Our Specific LP

- variables $d_x, d_y \to$ dimension 2
- one constraint for each face: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- $a_{i,j}, b_i, c_i$ are constants
- an objective function
- n constraints
- d is the dimension of the LP

$$
\begin{array}{rl}
\text{maximize} & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\
\text{such that} & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\
& a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\
& \vdots \\
& a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n
\end{array}
$$

Our Specific LP

- variables $d_x, d_y \to$ dimension 2
- one constraint for each face: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- no objective function

Linear Programs

General Form Of An LP

- variables $x_1, \dots, x_d$
- $a_{i,j}$, b_i, c_i are constants
- an objective function
- n constraints
- d is the dimension of the LP

$$\begin{aligned} \text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\ \text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\ & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\ & \vdots \\ & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n \end{aligned}$$

Our Specific LP

- variables $d_x, d_y \to$ dimension 2
- one constraint for each face: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- no objective function

Algorithm For The Mold Creation Problem

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- an objective function
- d is the dimension of the LP
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$\begin{aligned} \text{maximize} \quad & c_1 x_1 + c_2 x_2 + \cdots + c_d x_d \\ \text{such that} \quad & a_{1,1} x_1 + \cdots + a_{1,d} x_d \leq b_1 \\ & a_{2,1} x_1 + \cdots + a_{2,d} x_d \leq b_2 \\ & \vdots \\ & a_{n,1} x_1 + \cdots + a_{n,d} x_d \leq b_n \end{aligned}$$

Our Specific LP

- variables d_x, d_y $\to$ dimension 2
- one constraint for each face: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- no objective function

Algorithm For The Mold Creation Problem

- choose each of the n faces once as upper face
- for every upper face, solve a 2-dimensional LP with $n - 1$ constraints

Linear Programs

General Form Of An LP

- variables $x_1, \ldots, x_d$
- an objective function
- d is the dimension of the LP
- $a_{i,j}$, b_i, c_i are constants
- n constraints

$$\begin{aligned} \text{maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\ \text{such that} \quad & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leq b_1 \\ & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leq b_2 \\ & \vdots \\ & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leq b_n \end{aligned}$$

Our Specific LP

- variables $d_x, d_y \to$ dimension 2
- one constraint for each face: $\eta_x \cdot d_x + \eta_y \cdot d_y + \eta_z \geq 0$
- no objective function

Algorithm For The Mold Creation Problem

- choose each of the n faces once as upper face
- for every upper face, solve a 2-dimensional LP with $n - 1$ constraints

Goal In The Following: efficient algorithm to solve a 2-dimensional LP

Find An Optimal Solution

maximize: $x_1 + x_2$

such that:

$$
\begin{aligned}
x_1 &\geq 0 && (B)\\
x_2 &\geq 0 && (R)\\
x_2 - x_1 &\leq 1 && (E)\\
x_1 + 6x_2 &\leq 15 && (A)\\
4x_1 - x_2 &\leq 10 && (K)
\end{aligned}
$$

2D LPs

Example

$$
\begin{aligned}
\text{maximize:} \quad & x_1 + x_2 \\
\text{such that:} \quad & x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_2 - x_1 \leq 1 \\
& x_1 + 6x_2 \leq 15 \\
& 4x_1 - x_2 \leq 10
\end{aligned}
$$

2D LPs

Example

maximize: $x_1 + x_2$

such that:
$$
\begin{aligned}
x_1 &\geq 0\\
x_2 &\geq 0\\
x_2 - x_1 &\leq 1\\
x_1 + 6x_2 &\leq 15\\
4x_1 - x_2 &\leq 10
\end{aligned}
$$

Geometric Interpretation

2D LPs

Example

maximize: $x_1 + x_2$

such that: $x_1 \geq 0$

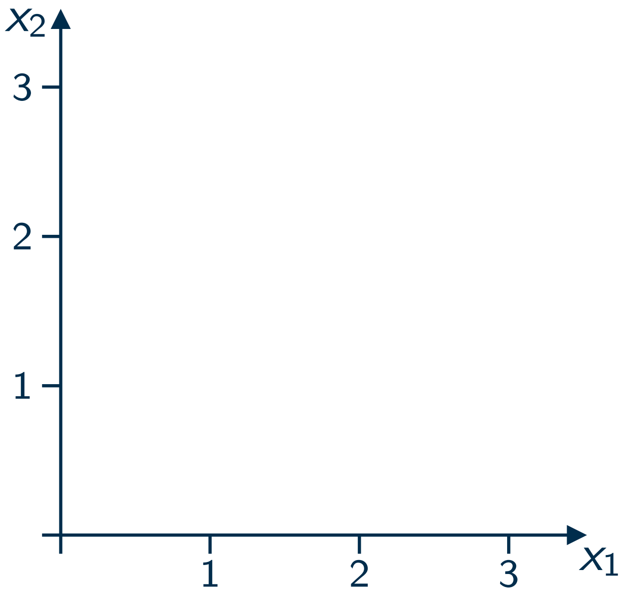
$x_2 \geq 0$

$x_2 - x_1 \leq 1$

$x_1 + 6x_2 \leq 15$

$4x_1 - x_2 \leq 10$

Geometric Interpretation



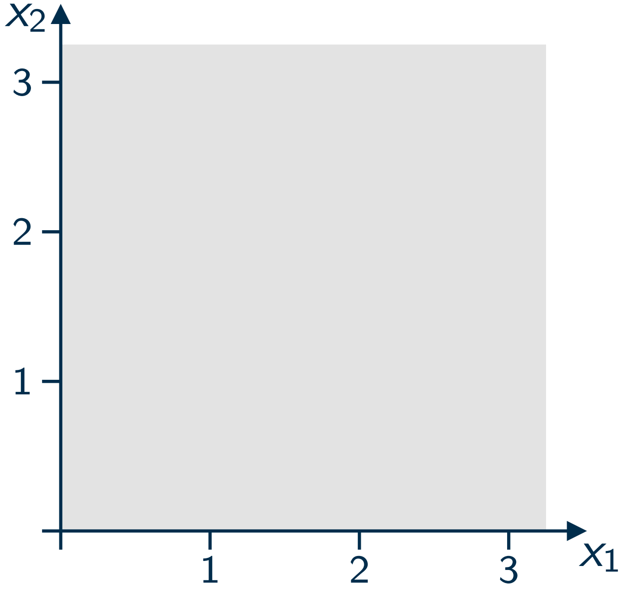
2D LPs

Example

maximize: $x_1 + x_2$

such that:
$$
\begin{aligned}
x_1 &\geq 0\\
x_2 &\geq 0\\
x_2 - x_1 &\leq 1\\
x_1 + 6x_2 &\leq 15\\
4x_1 - x_2 &\leq 10
\end{aligned}
$$

Geometric Interpretation



2D LPs

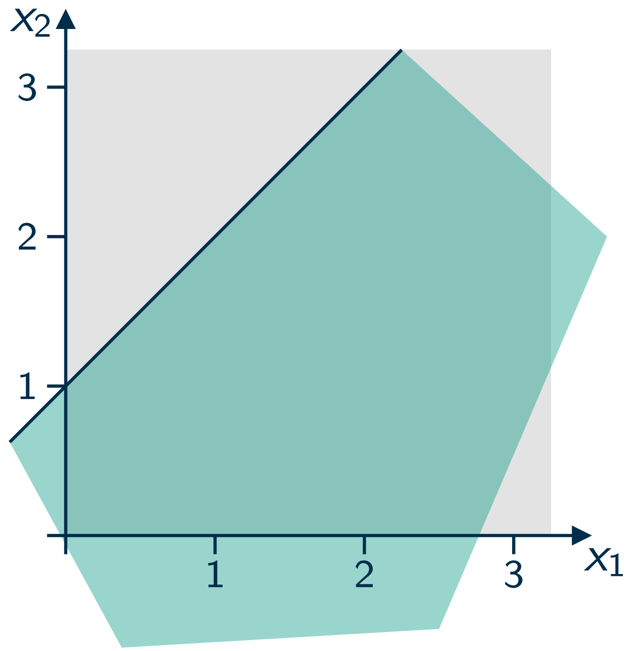
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



2D LPs

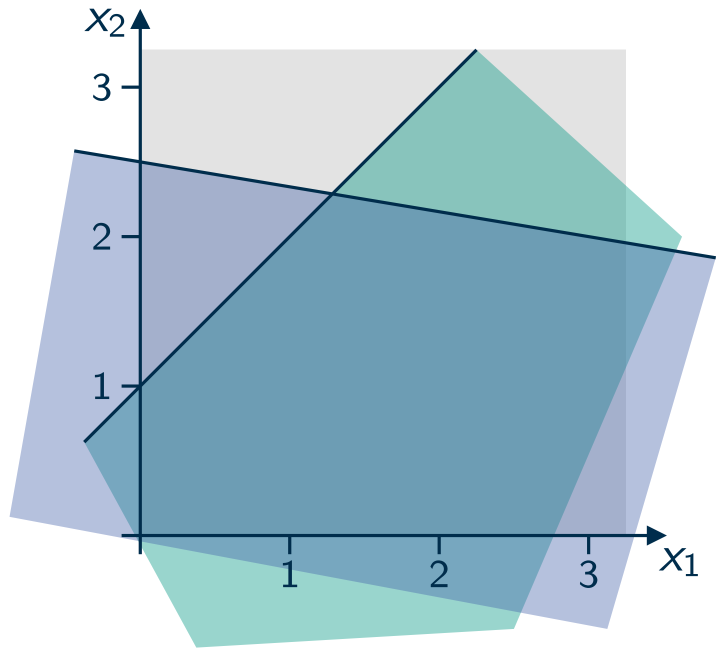
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



2D LPs

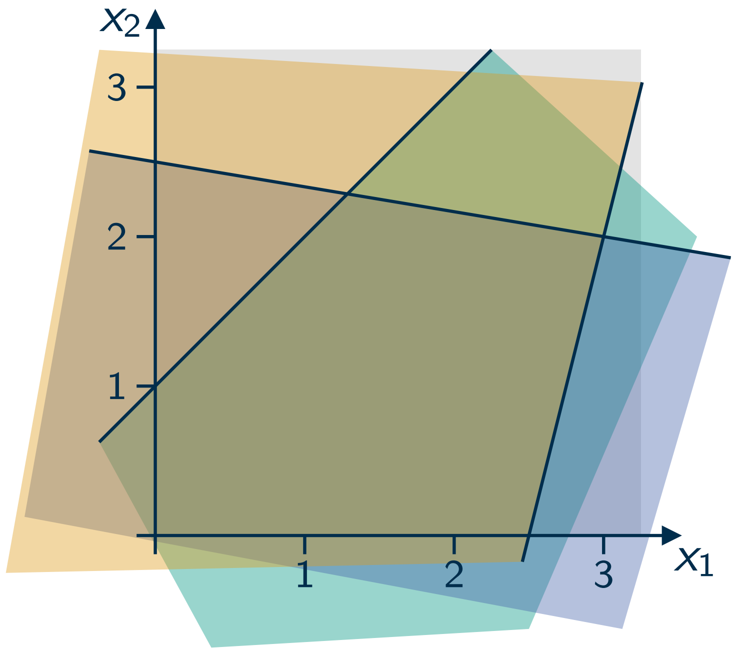
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



2D LPs

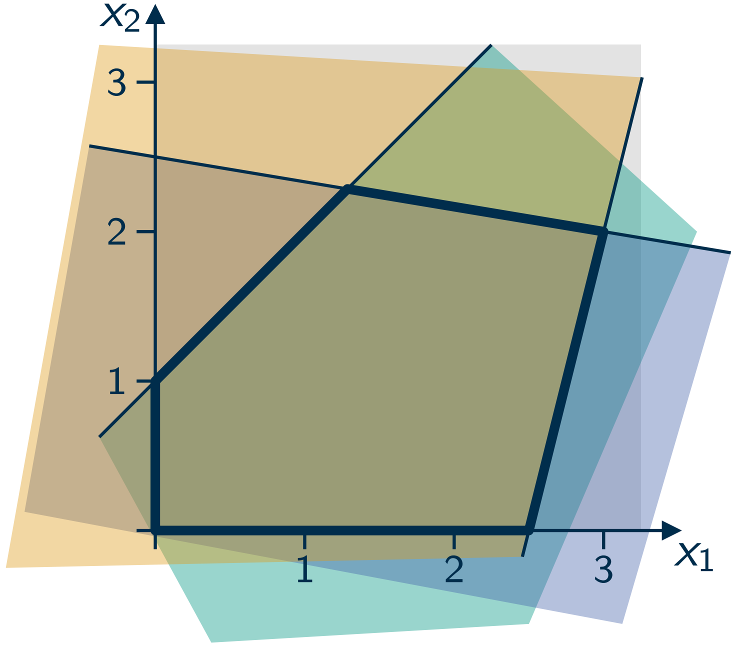
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



2D LPs

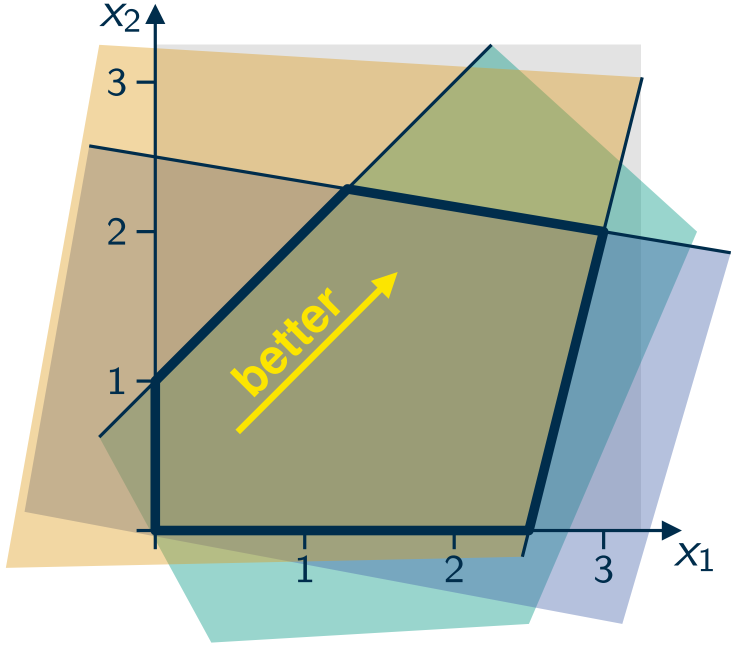
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



2D LPs

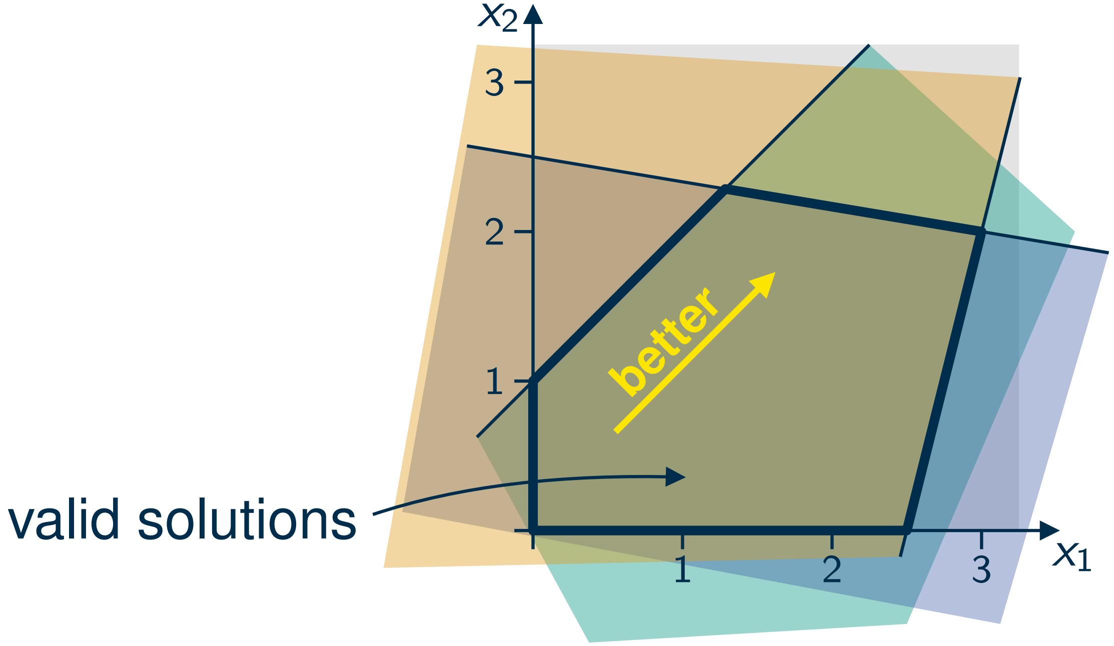
Example

maximize: $x_1 + x_2$

such that:

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_2 - x_1 \leq 1$$
$$x_1 + 6x_2 \leq 15$$
$$4x_1 - x_2 \leq 10$$

Geometric Interpretation



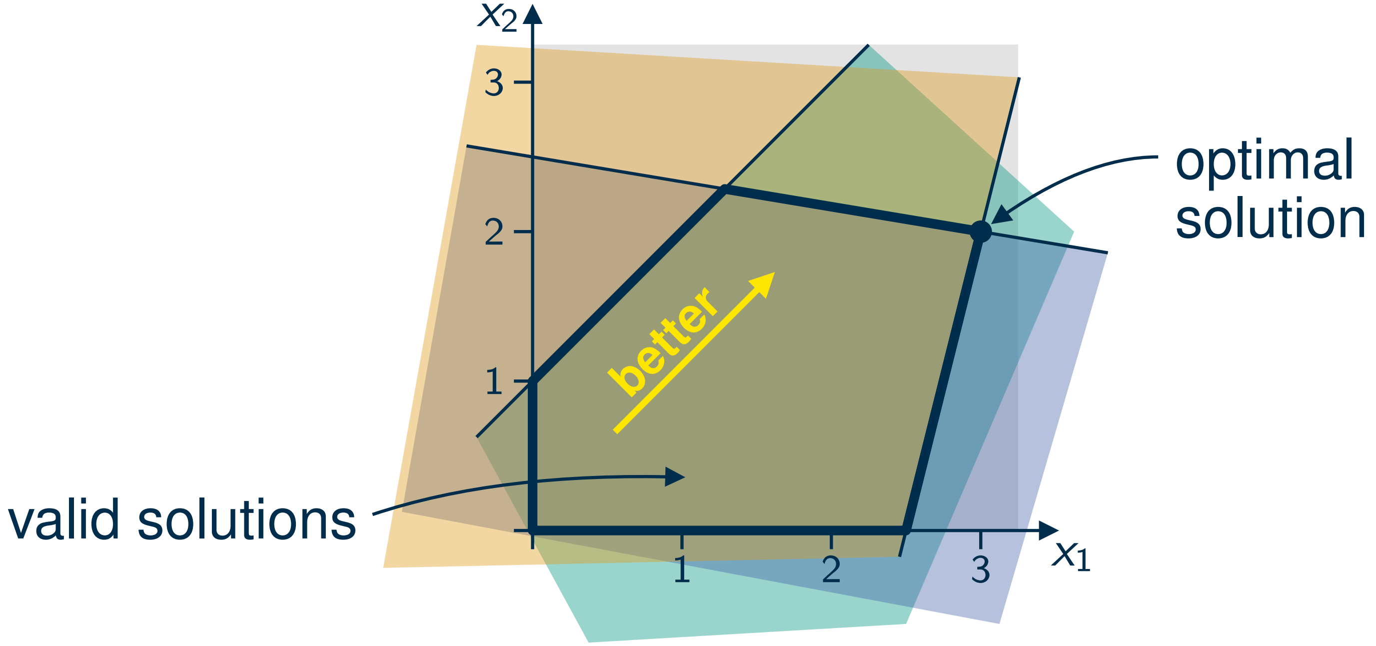
2D LPs

Example

maximize: $x_1 + x_2$

such that:
$x_1 \geq 0$
$x_2 \geq 0$
$x_2 - x_1 \leq 1$
$x_1 + 6x_2 \leq 15$
$4x_1 - x_2 \leq 10$

Geometric Interpretation



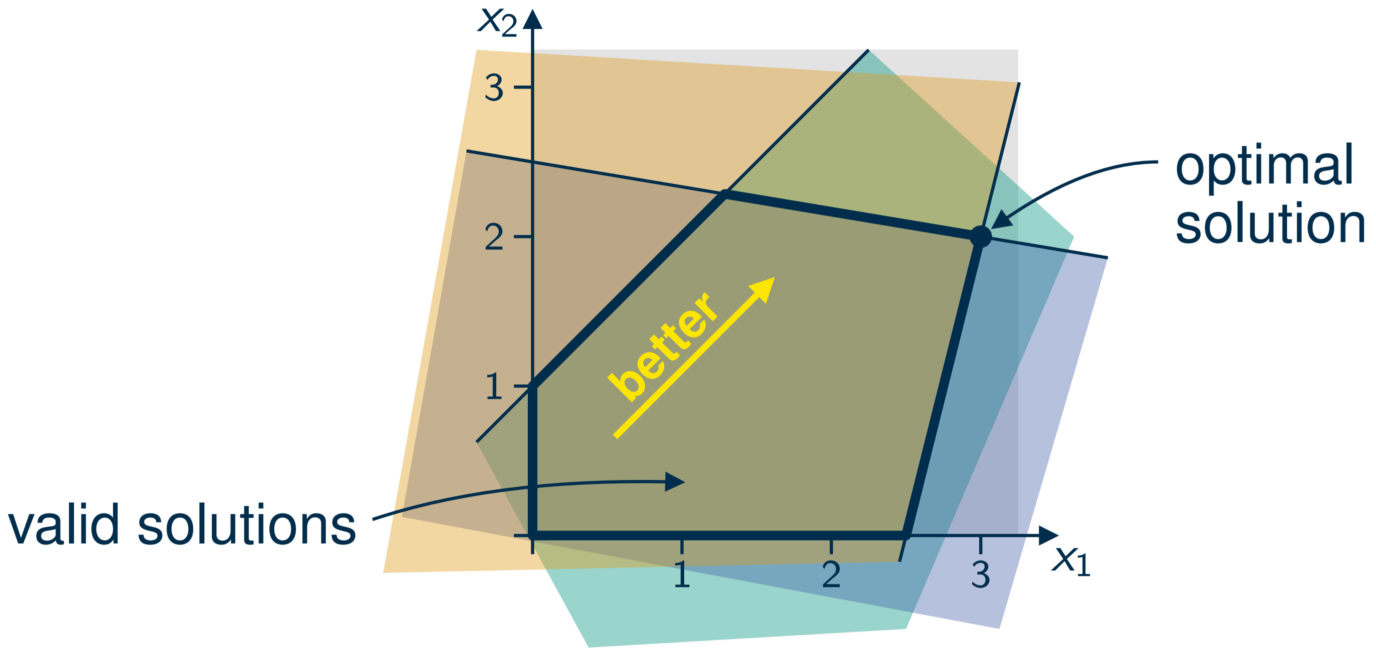
2D LPs

Example

maximize: $x_1 + x_2$

such that:
$x_1 \geq 0$
$x_2 \geq 0$
$x_2 - x_1 \leq 1$
$x_1 + 6x_2 \leq 15$
$4x_1 - x_2 \leq 10$

Geometric Interpretation



Properties Of The LP

- **infeasible:** no valid solution

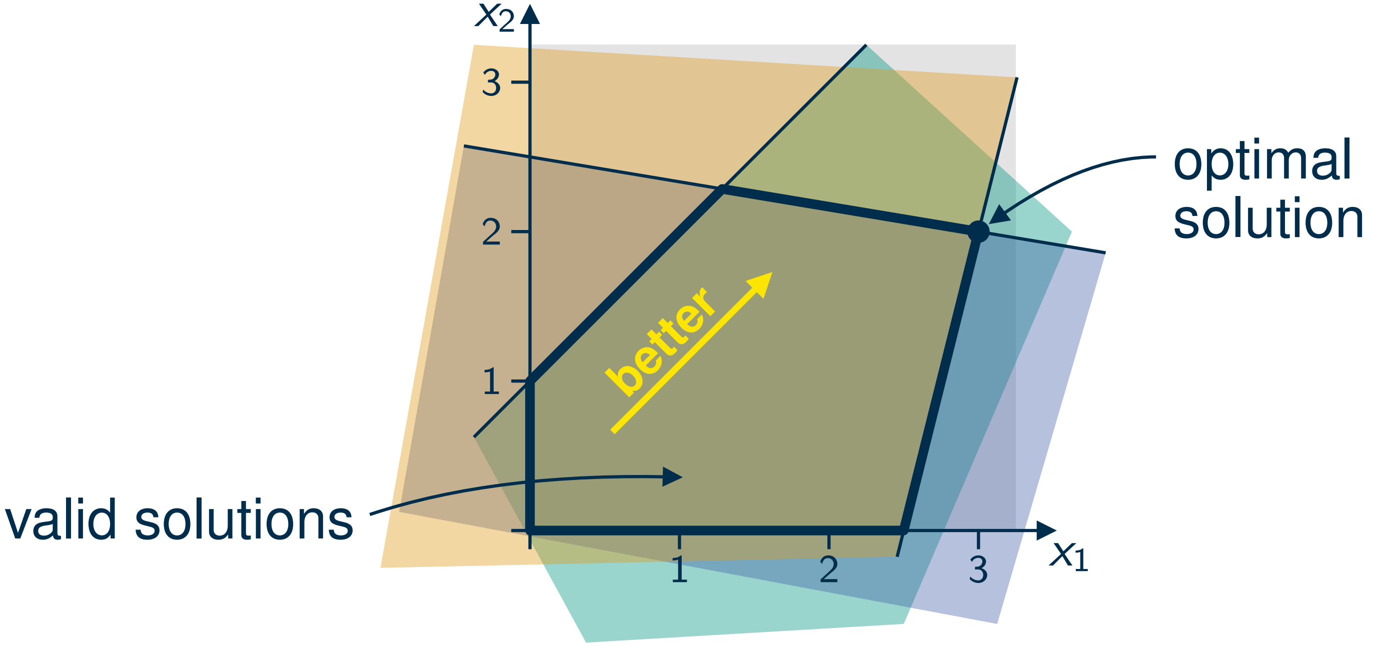
2D LPs

Example

maximize: $x_1 + x_2$

such that:
$x_1 \geq 0$
$x_2 \geq 0$
$x_2 - x_1 \leq 1$
$x_1 + 6x_2 \leq 15$
$4x_1 - x_2 \leq 10$

Geometric Interpretation



Properties Of The LP

- **infeasible:** no valid solution
- **unbounded:** there are solutions with arbitrarily large objective

2D LPs

Example

maximize: $x_1 + x_2$

such that:

$x_1 \geq 0$

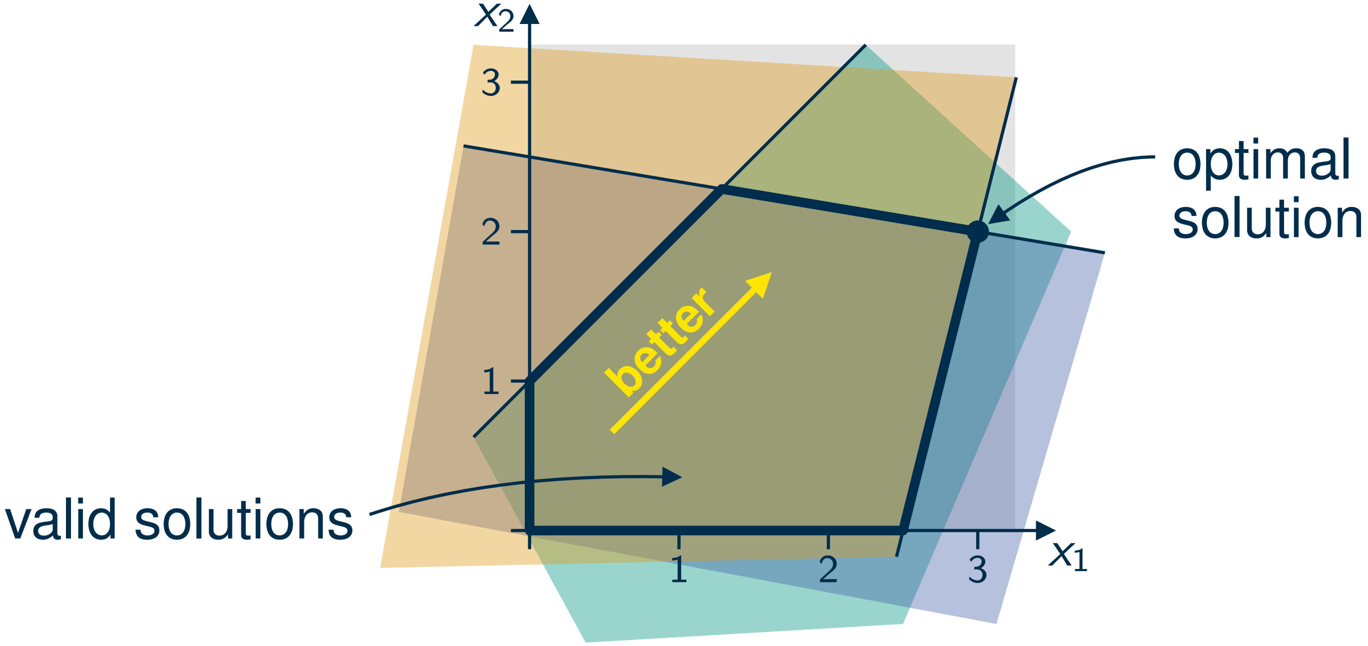
$x_2 \geq 0$

$x_2 - x_1 \leq 1$

$x_1 + 6x_2 \leq 15$

$4x_1 - x_2 \leq 10$

Geometric Interpretation



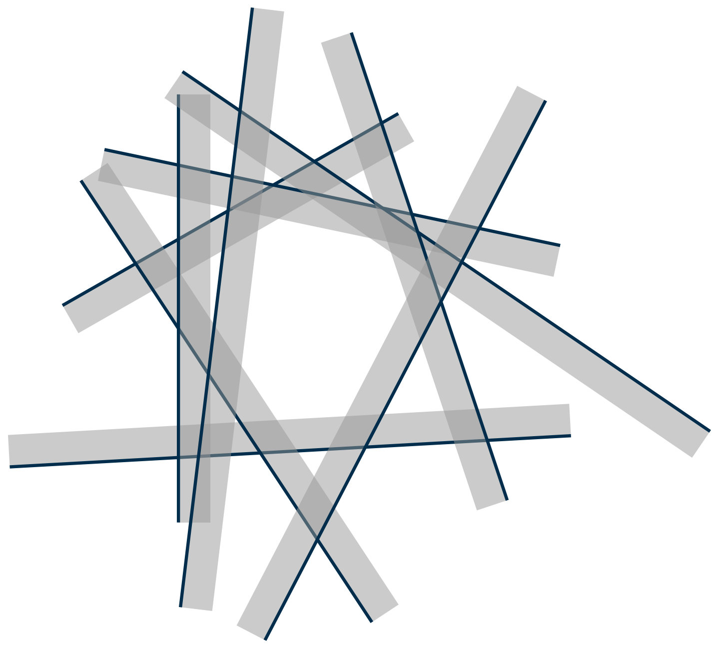
Properties Of The LP

- **infeasible:** no valid solution
- **unbounded:** there are solutions with arbitrarily large objective

Problem: Half-Plane Intersection
Given n half planes, compute their intersection.

Half-Plane Intersection

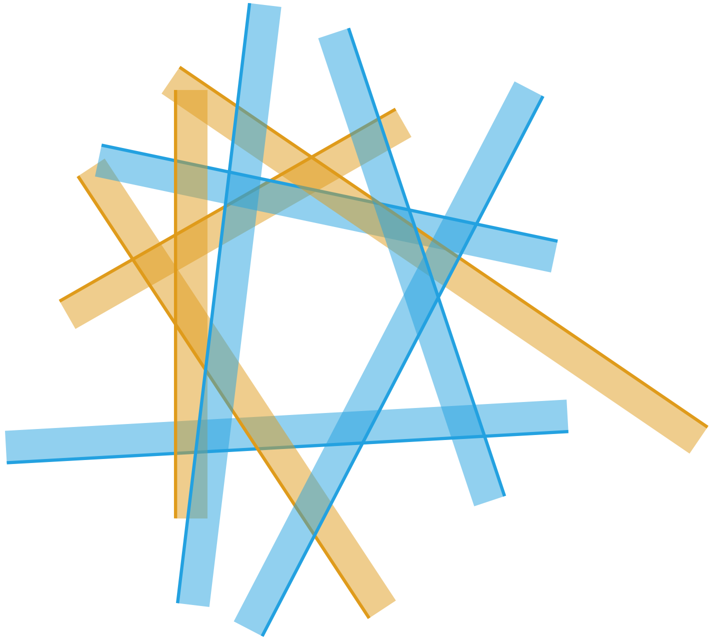
Plan: Divide And Conquer



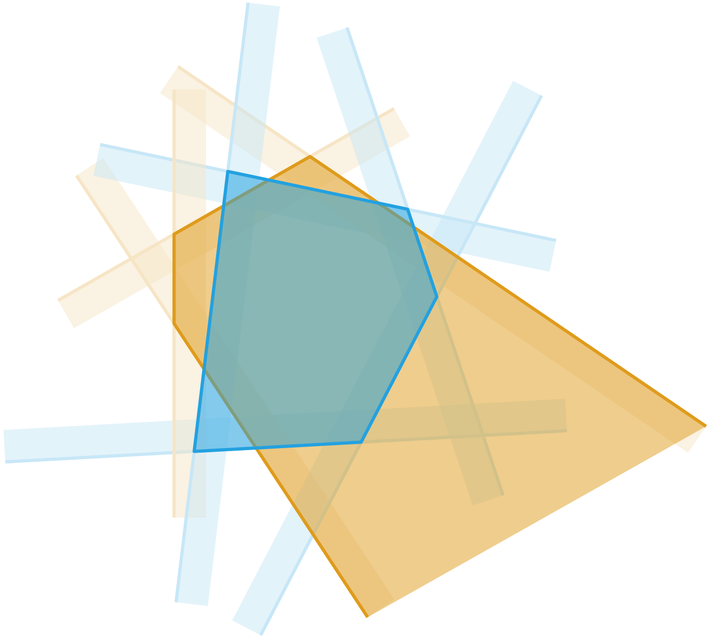
Half-Plane Intersection

Plan: Divide And Conquer

- split half planes into two groups of roughly equal size



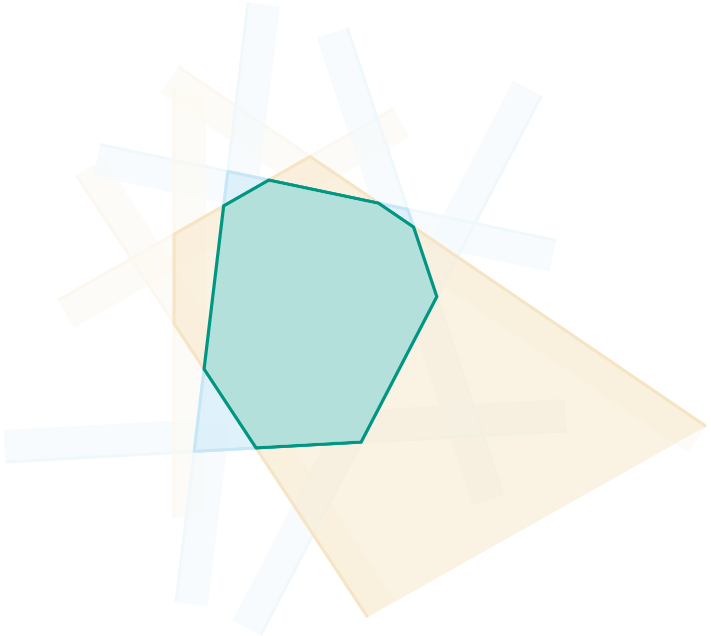
Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group

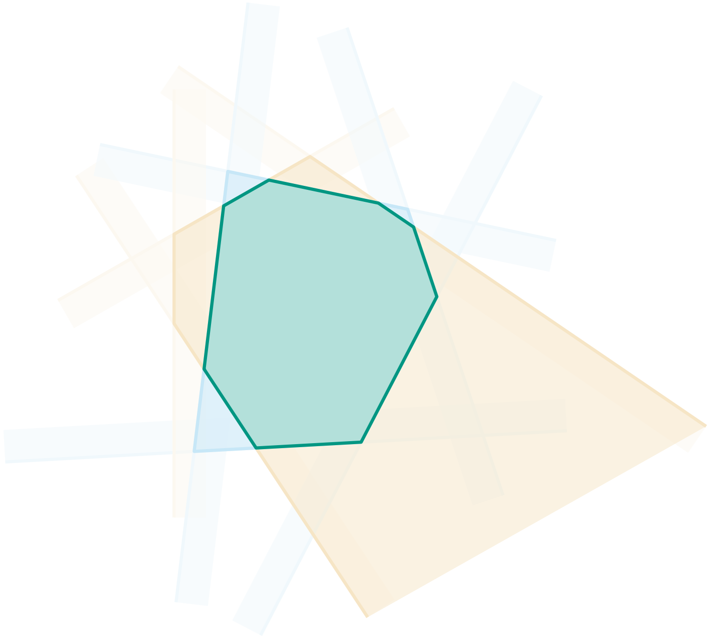
Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Half-Plane Intersection

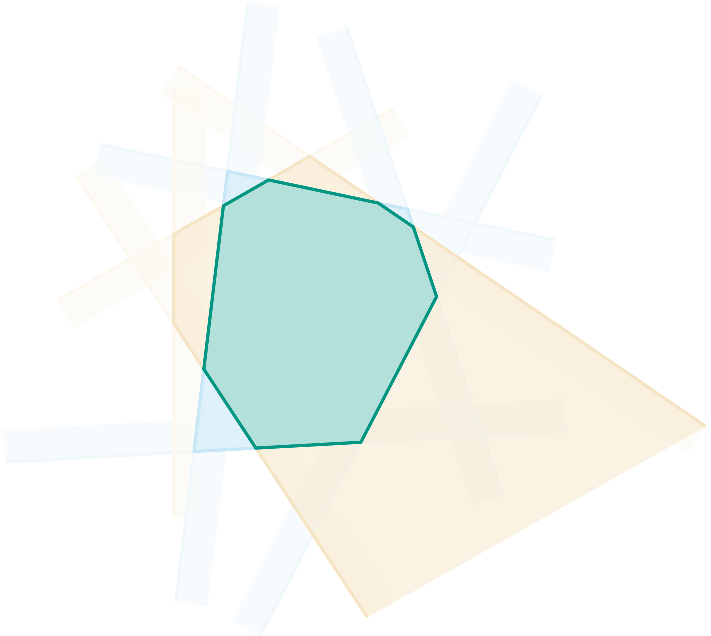


Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

Half-Plane Intersection



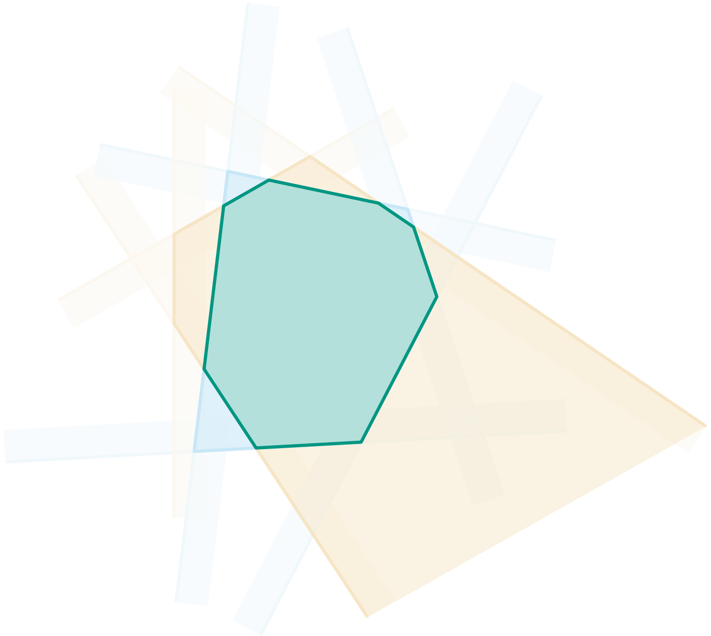
Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

- more or less the intersection of two convex polygons

Half-Plane Intersection



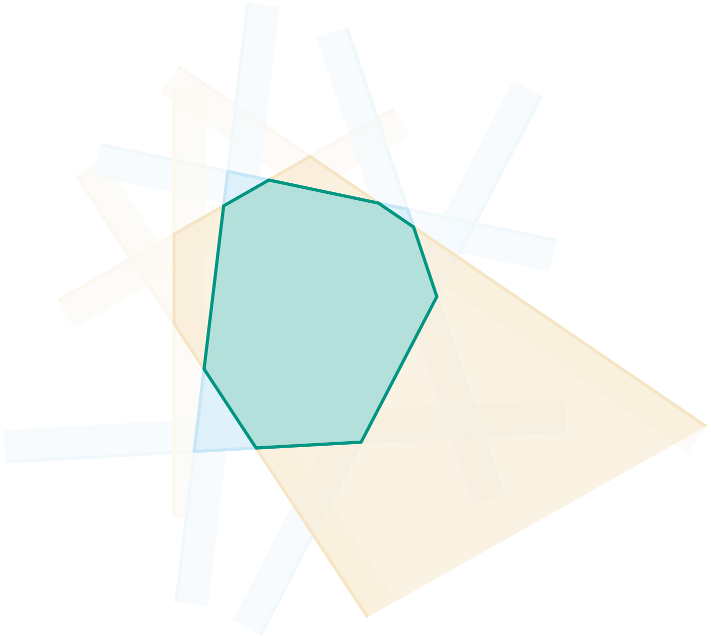
Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded

Half-Plane Intersection



Plan: Divide And Conquer

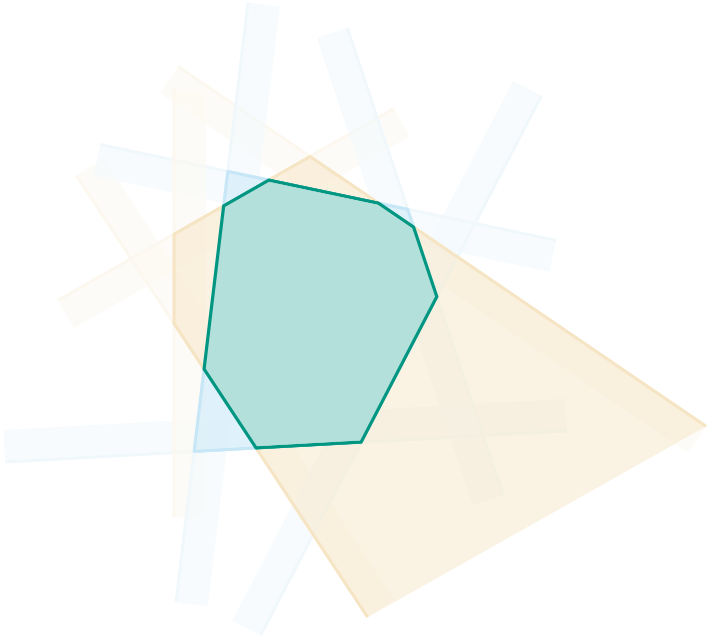
- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$

this is a hint for one exercise on the current sheet

Half-Plane Intersection



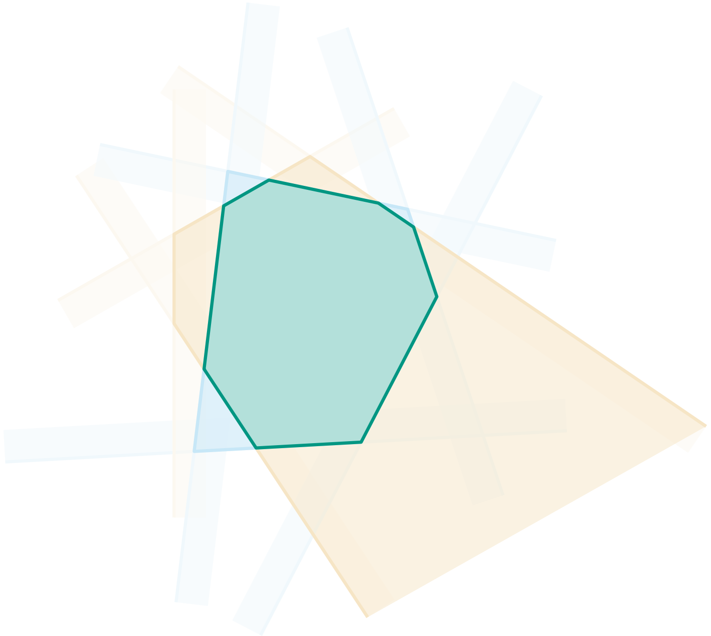
Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$
  this is a hint for one exercise on the current sheet
- using convexity $\rightarrow$ running time $O(n)$

Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

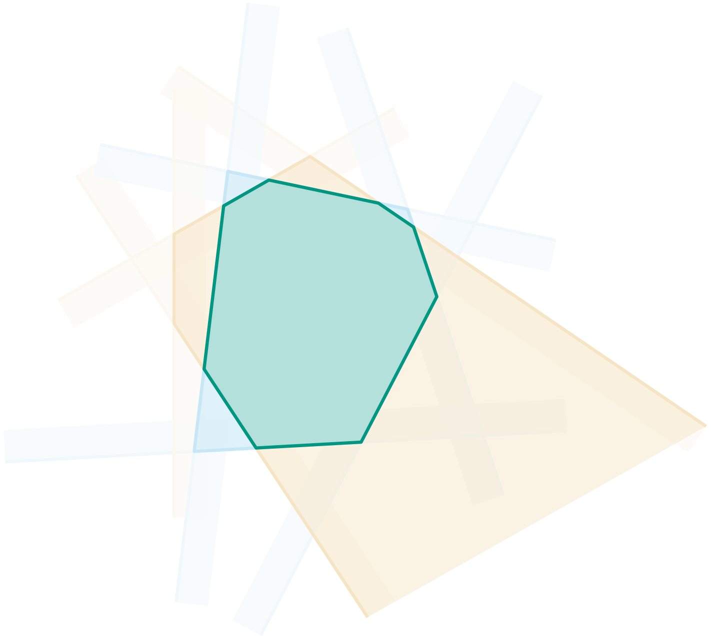
Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$
- using convexity $\rightarrow$ running time $O(n)$

this is a hint for one exercise on the current sheet

Total Time:

Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

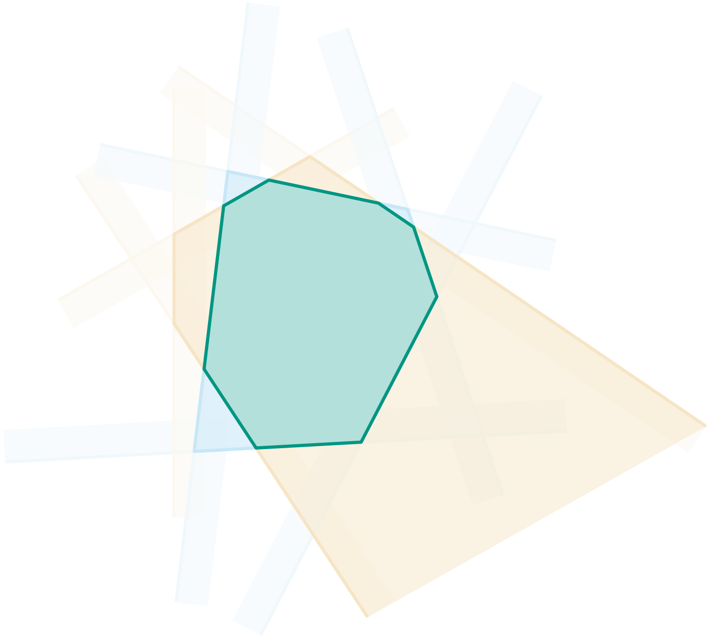
Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$
- using convexity $\rightarrow$ running time $O(n)$

this is a hint for one exercise on the current sheet

Total Time: $T(n) = O(n) + 2T(n/2)$

Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

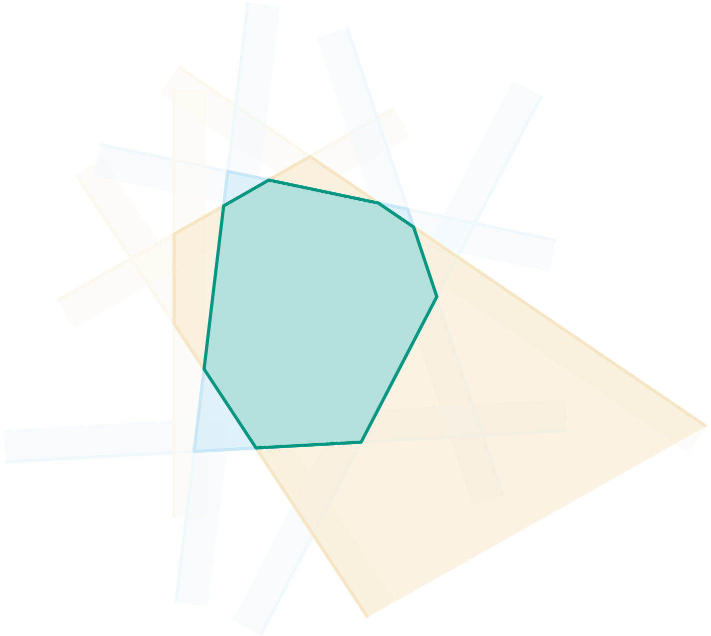
- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$
- using convexity $\rightarrow$ running time $O(n)$

this is a hint for one exercise on the current sheet

Total Time: $T(n) = O(n) + 2T(n/2)$

$\Rightarrow O(n \log n)$

Half-Plane Intersection



Plan: Divide And Conquer

- split half planes into two groups of roughly equal size
- compute intersection for each group
- compute intersection of the two resulting regions

Intersecting The Two Results

- more or less the intersection of two convex polygons
- careful: regions might be unbounded
- sweep-line algorithm can be adjusted accordingly $\rightarrow$ running time $O(n \log n)$
- using convexity $\rightarrow$ running time $O(n)$

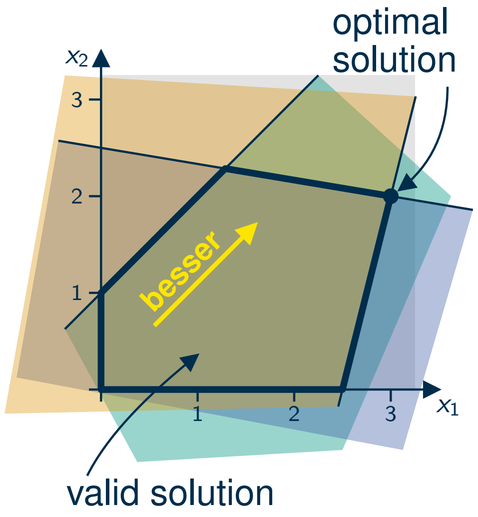
this is a hint for one exercise on the current sheet

Total Time: $T(n) = O(n) + 2T(n/2)$
$\Rightarrow O(n \log n)$

Can This Be Improved?

- closely related to convex hull (via duality)
- lower bound: $\Omega(n \log n)$

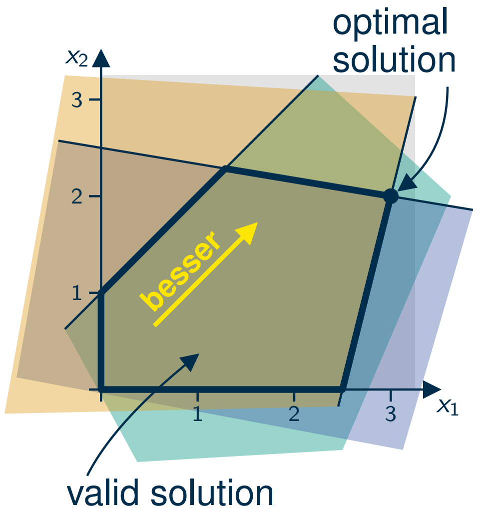
Incremental Algorithm For 2D LPs



Observation

- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Algorithm For 2D LPs



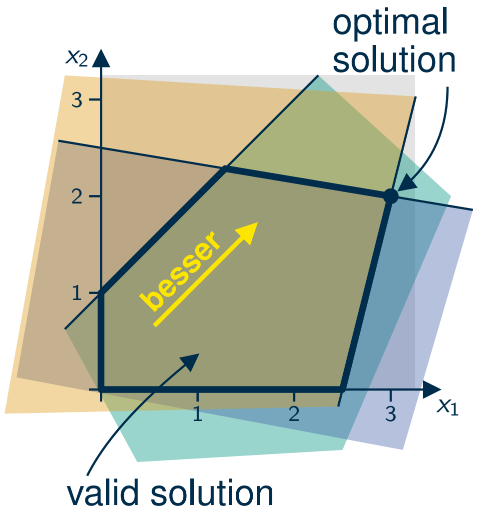
Observation

- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

- let $h_1, \dots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \dots \cap h_i$ (feasible region with respect to $h_1, \dots, h_i$)

Incremental Algorithm For 2D LPs



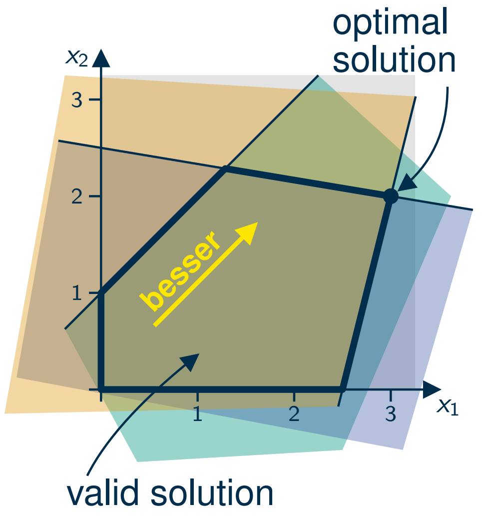
Observation

- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

- let $h_1, \ldots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \cdots \cap h_i$ (feasible region with respect to $h_1, \ldots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$

Incremental Algorithm For 2D LPs



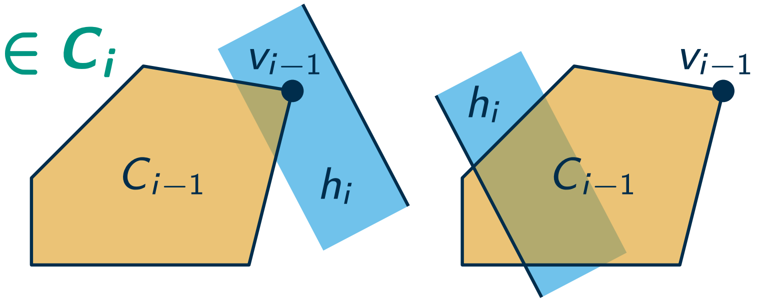
Observation

- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

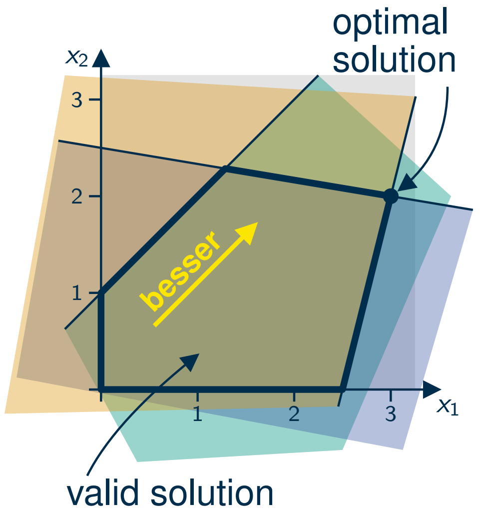
Incremental Approach

- let $h_1, \ldots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \cdots \cap h_i$ (feasible region with respect to $h_1, \ldots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$

Case 1: $v_{i-1} \in C_i$



Incremental Algorithm For 2D LPs



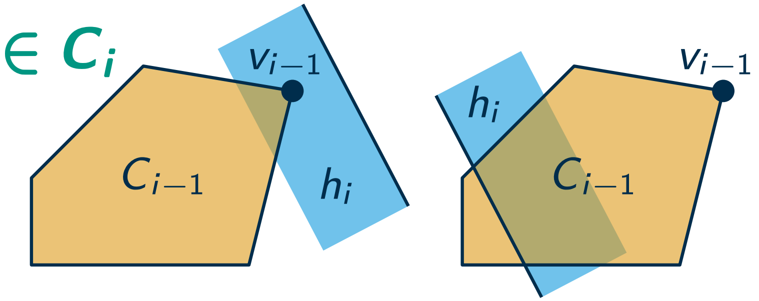
Observation

- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

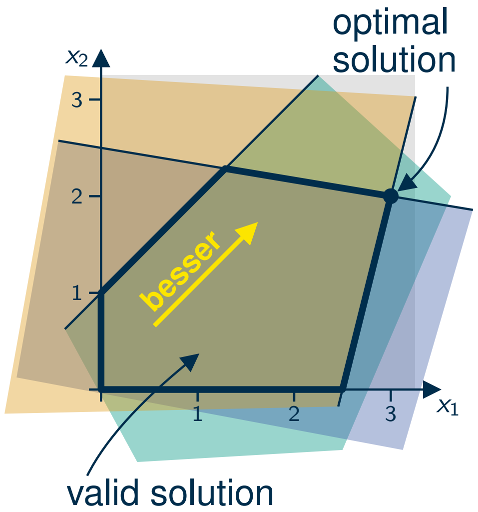
- let $h_1, \ldots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \cdots \cap h_i$ (feasible region with respect to $h_1, \ldots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$

Case 1: $v_{i-1} \in C_i$



- v_{i-1} remains optimal for C_i (as $C_i \subseteq C_{i-1}$)

Incremental Algorithm For 2D LPs

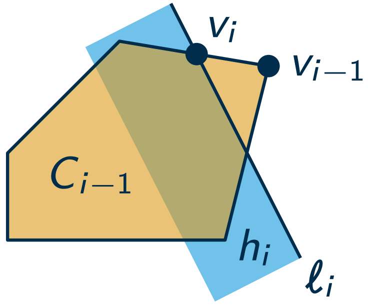


Observation

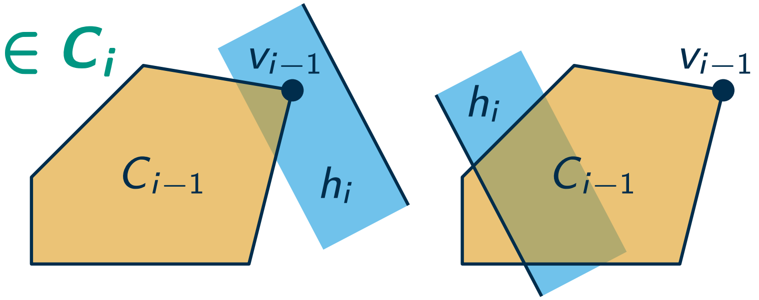
- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

- let $h_1, \ldots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \cdots \cap h_i$ (feasible region with respect to $h_1, \ldots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$



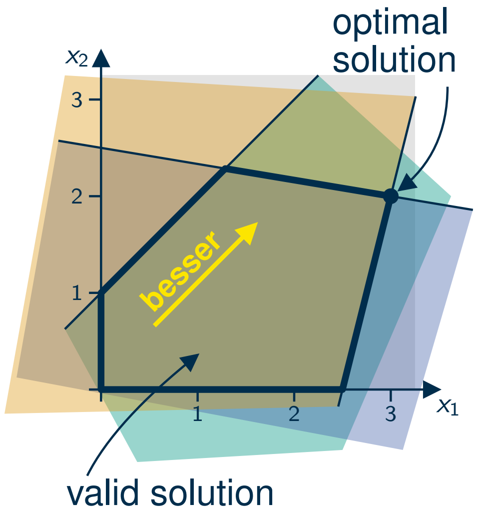
Case 1: $v_{i-1} \in C_i$



Case 2: $v_{i-1} \notin C_i$

- v_{i-1} remains optimal for C_i (as $C_i \subseteq C_{i-1}$)

Incremental Algorithm For 2D LPs



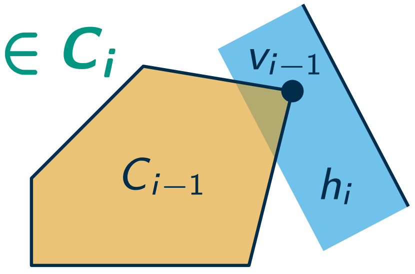
Observation

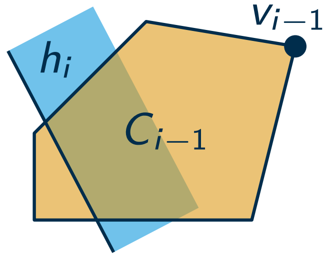
- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

- let $h_1, \dots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \dots \cap h_i$ (feasible region with respect to $h_1, \dots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$

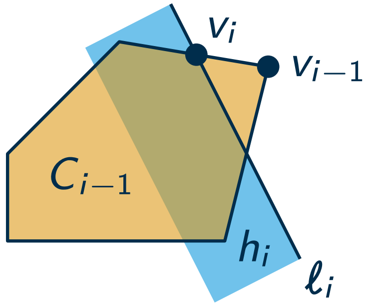
Case 1: $v_{i-1} \in C_i$





- v_{i-1} remains optimal for C_i (as $C_i \subseteq C_{i-1}$)

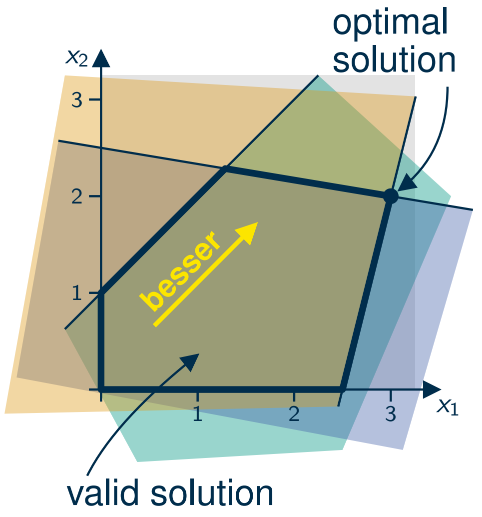
Case 2: $v_{i-1} \notin C_i$



- v_i lies on the line ℓ_i bounding h_i

Why?

Incremental Algorithm For 2D LPs

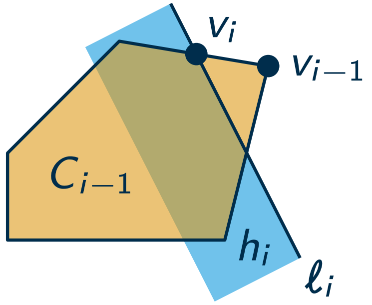


Observation

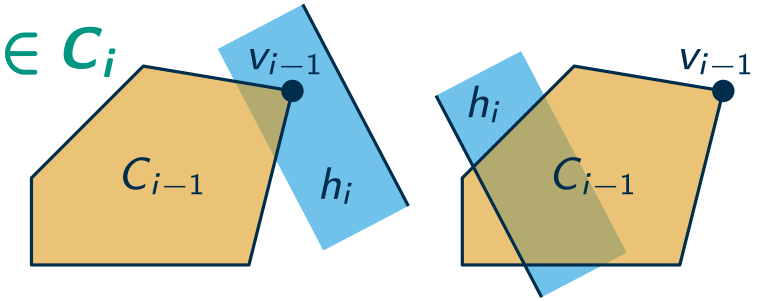
- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

Incremental Approach

- let $h_1, \dots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \dots \cap h_i$ (feasible region with respect to $h_1, \dots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$



Case 1: $v_{i-1} \in C_i$



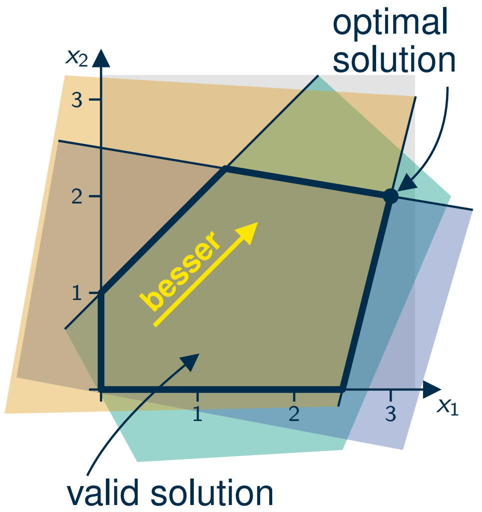
- v_{i-1} remains optimal for C_i (as $C_i \subseteq C_{i-1}$)

Case 2: $v_{i-1} \notin C_i$

- v_i lies on the line ℓ_i bounding h_i
- this is essentially a 1D LP with i constraints

Why?

Incremental Algorithm For 2D LPs

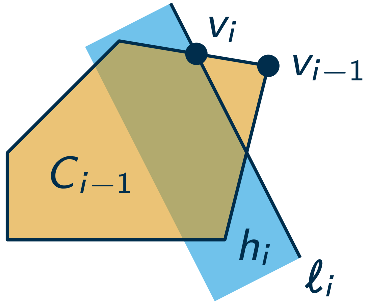


Observation

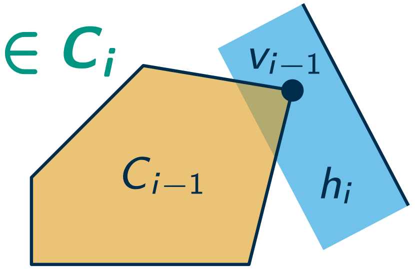
- we do not actually need to compute the valid region explicitly
- it is sufficient to compute a valid point that maximizes the objective

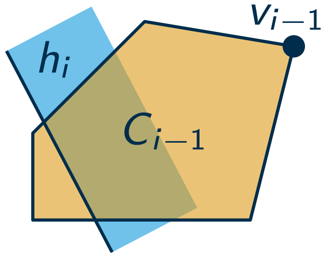
Incremental Approach

- let $h_1, \dots, h_n$ he the half planes (constraints)
- let $C_i = h_1 \cap h_2 \cap \cdots \cap h_i$ (feasible region with respect to $h_1, \dots, h_i$)
- assumption: we know an optimal point $v_{i-1} \in C_{i-1}$
- goal: find an optimal point $v_i \in C_i$



Case 1: $v_{i-1} \in C_i$





- v_{i-1} remains optimal for C_i (as $C_i \subseteq C_{i-1}$)

Case 2: $v_{i-1} \notin C_i$

Why?

- v_i lies on the line ℓ_i bounding h_i
- this is essentially a 1D LP with i constraints
- can be solved in $O(i)$ time

How?

Incremental Algorithm For 2D LPs

Open Questions

- How do we start?
- Why do we care about an $O(n^2)$ algorithm?

Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

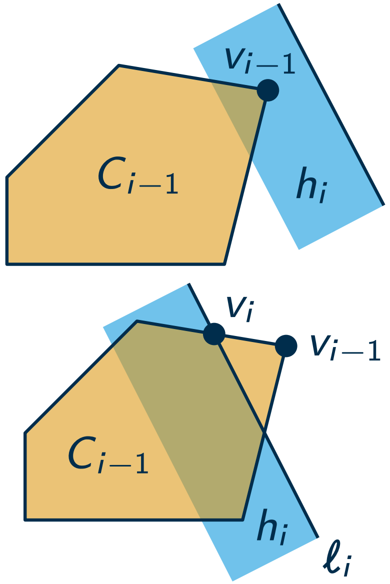
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)



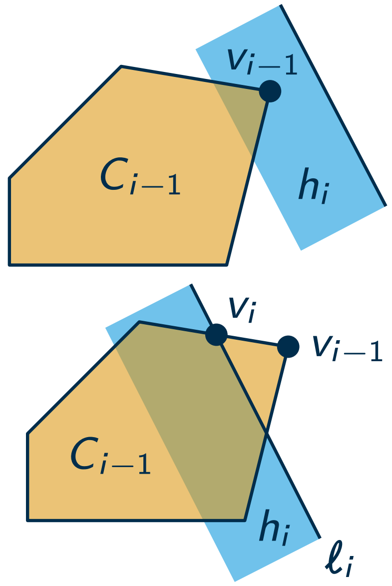
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)
- hope: $v_{i-1} \notin C_i$ happens rarely



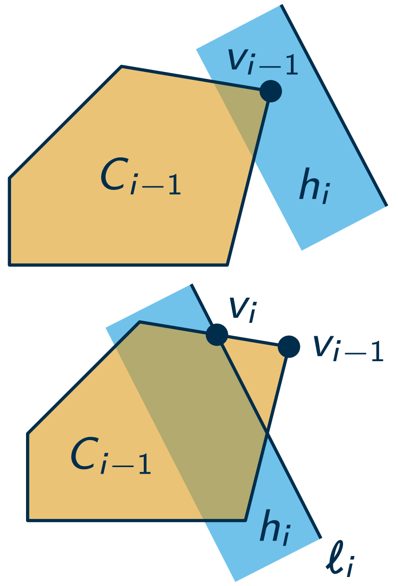
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)
- hope: $v_{i-1} \notin C_i$ happens rarely
- there is an order $h_1, \dots, h_n$, such that $v_{i-1} \in C_i$ for $i \geq 3$ Why?



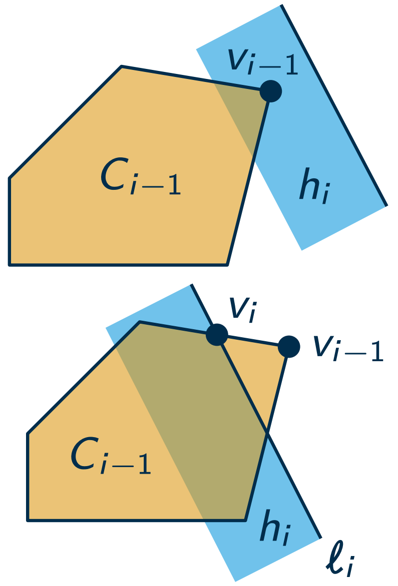
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)
- hope: $v_{i-1} \notin C_i$ happens rarely
- there is an order $h_1, \ldots, h_n$, such that $v_{i-1} \in C_i$ for $i \geq 3$ **Why?**
- finding this order is not so easy



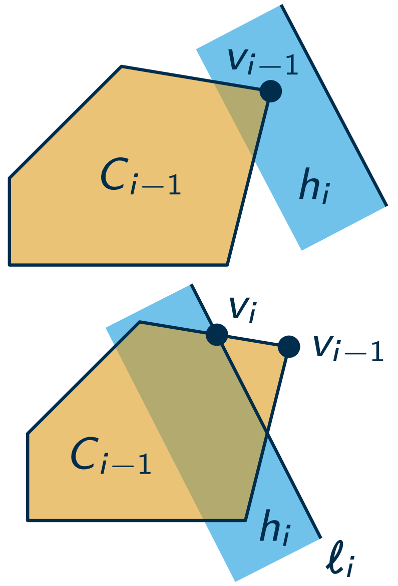
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\rightarrow$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\rightarrow$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)
- hope: $v_{i-1} \notin C_i$ happens rarely
- there is an order $h_1, \ldots, h_n$, such that $v_{i-1} \in C_i$ for $i \geq 3$ Why?
- finding this order is not so easy
- but: most orders are good



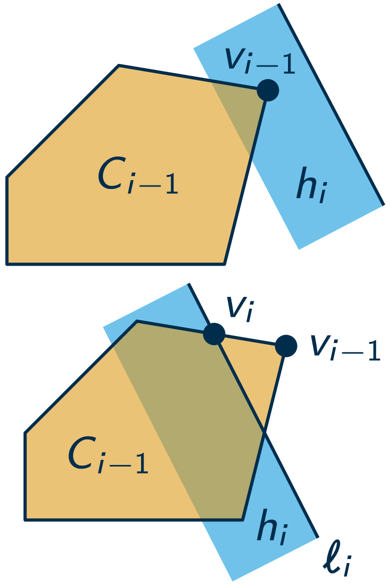
Incremental Algorithm For 2D LPs

Open Questions

- How do we start? $\to$ next exercise sheet
- Why do we care about an $O(n^2)$ algorithm? $\to$ now

Thoughts On The Running Time

- case $v_{i-1} \in C_i$ is cheap (just set $v_i = v_{i-1}$)
- case $v_{i-1} \notin C_i$ is expensive ($O(i)$ to compute v_i)
- hope: $v_{i-1} \notin C_i$ happens rarely
- there is an order $h_1, \dots, h_n$, such that $v_{i-1} \in C_i$ for $i \geq 3$ Why?
- finding this order is not so easy
- but: most orders are good $\to$ random order



Randomized Incremental Algorithm

Expected Running Time

- the running time is a random variable, let's call it X

Randomized Incremental Algorithm

Expected Running Time

- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i$$

Randomized Incremental Algorithm

Expected Running Time

- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right]$$

Randomized Incremental Algorithm

Expected Running Time

- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Randomized Incremental Algorithm

Expected Running Time

- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration *i*:

$\mathbb{E}[X_i] =$

Randomized Incremental Algorithm

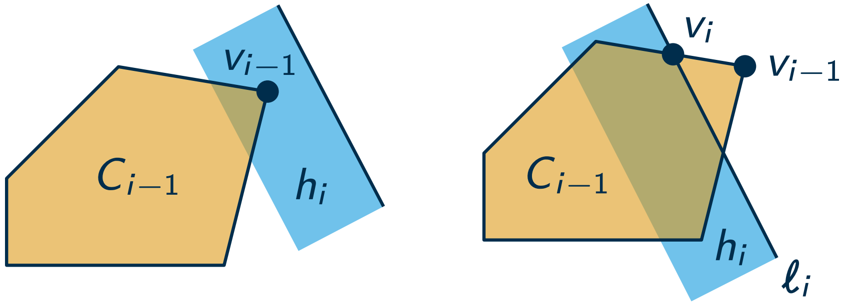
Expected Running Time

- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration *i*:

$\mathbb{E}[X_i] =$



Randomized Incremental Algorithm

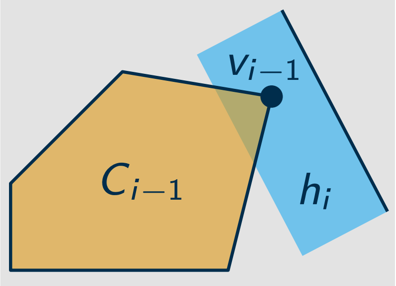
Expected Running Time

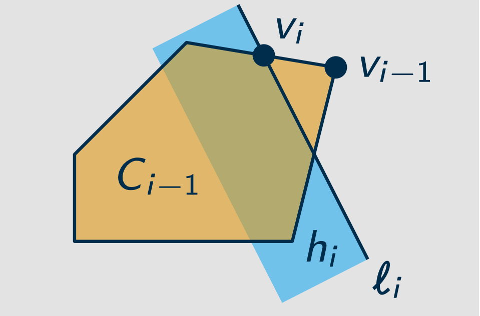
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Randomized Incremental Algorithm

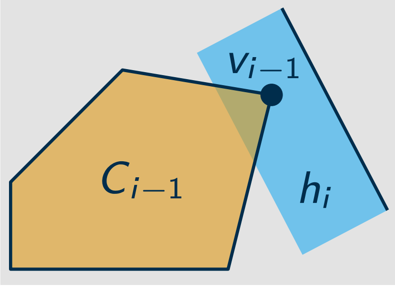
Expected Running Time

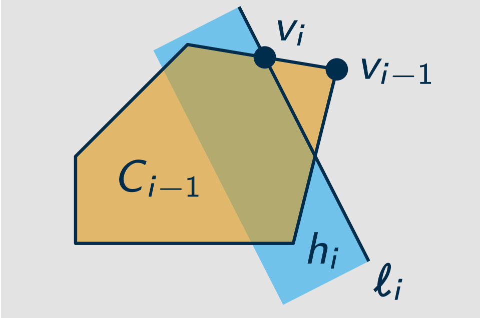
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

Randomized Incremental Algorithm

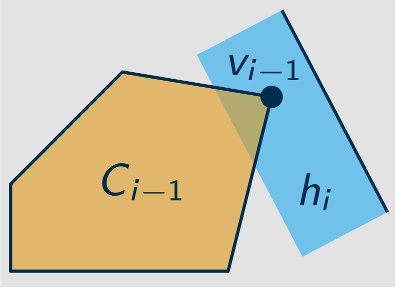
Expected Running Time

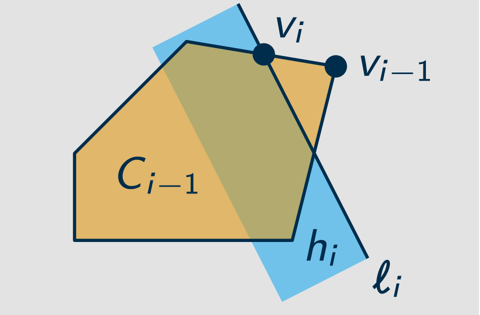
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order

Randomized Incremental Algorithm

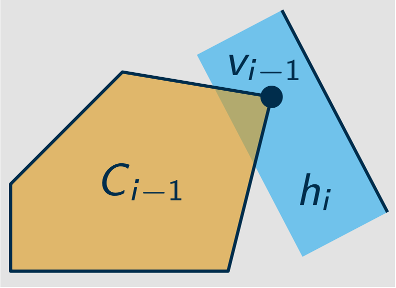
Expected Running Time

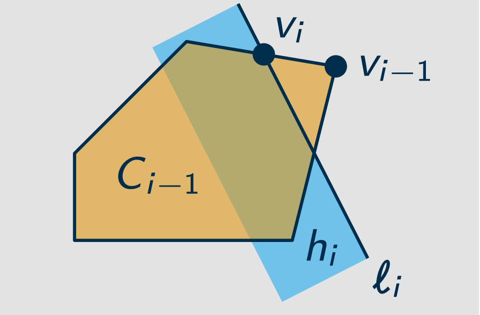
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step

Randomized Incremental Algorithm

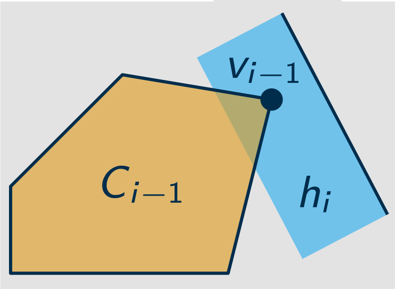
Expected Running Time

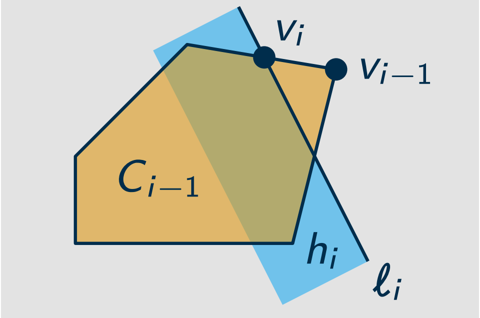
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

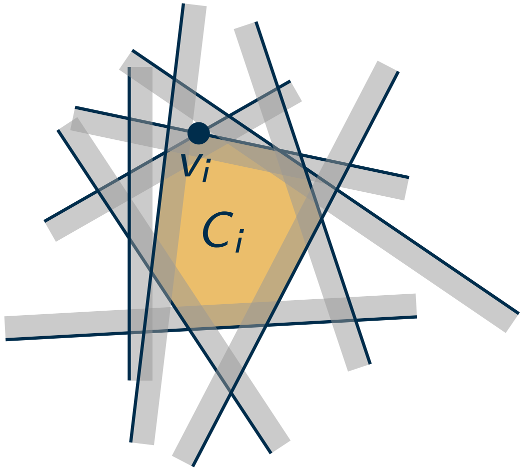
$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)



Randomized Incremental Algorithm

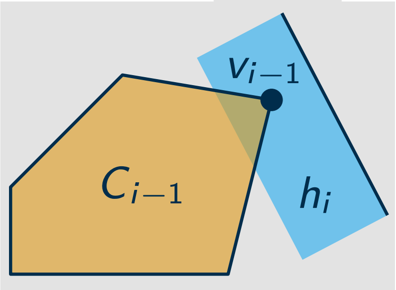
Expected Running Time

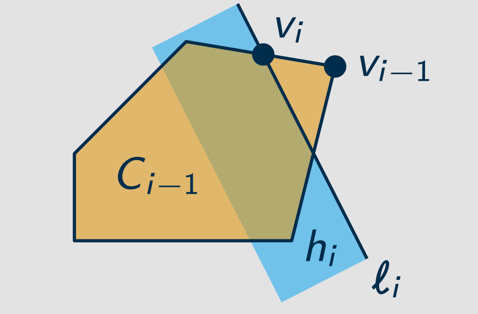
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

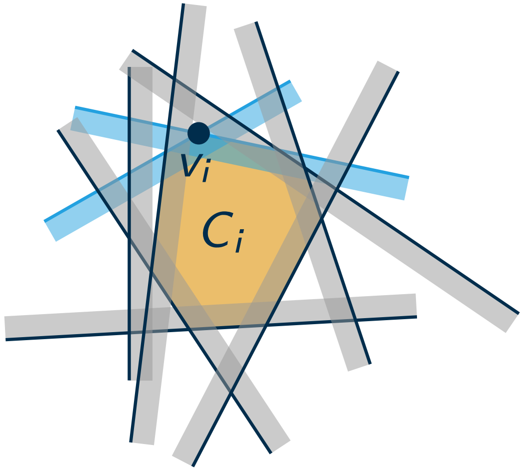
$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i



Randomized Incremental Algorithm

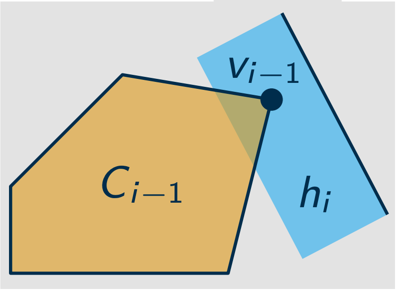
Expected Running Time

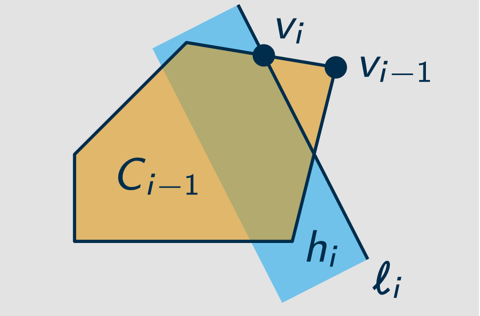
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

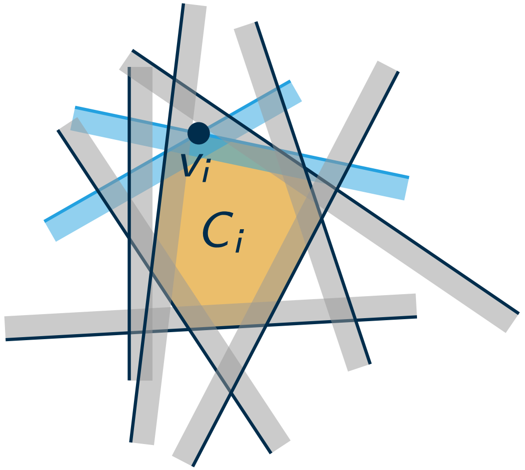
$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i
  - probability: $\frac{2}{i}$



Randomized Incremental Algorithm

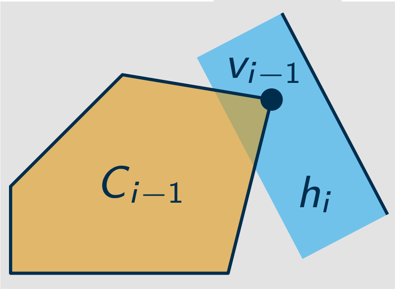
Expected Running Time

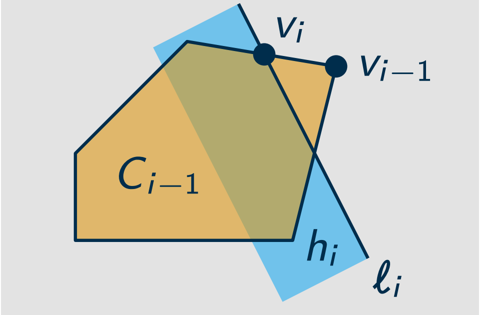
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

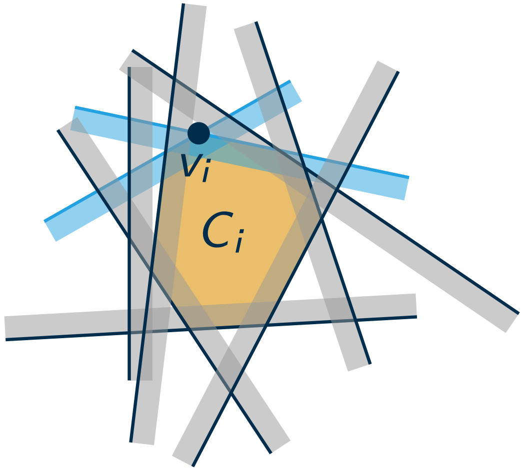
$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$





Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i
  - probability: $\frac{2}{i}$
- thus: $P(v_i \neq v_{i-1}) \leq \frac{2}{i}$



Randomized Incremental Algorithm

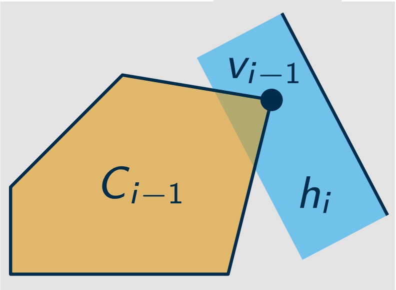
Expected Running Time

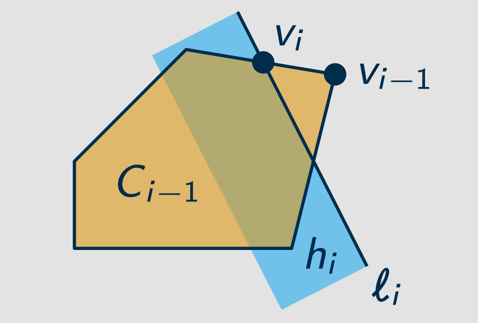
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$

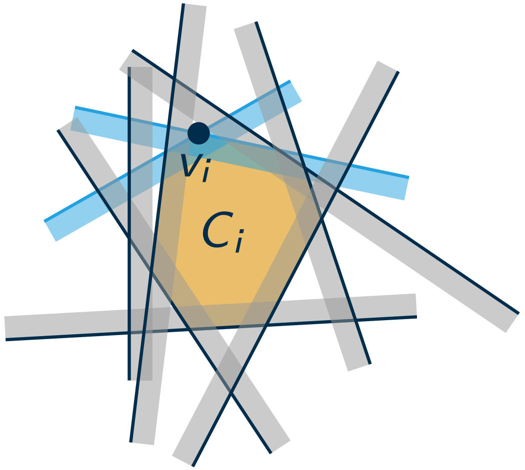




Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i
  - probability: $\frac{2}{i}$
- thus: $P(v_i \neq v_{i-1}) \leq \frac{2}{i}$

Why $\leq$?



Randomized Incremental Algorithm

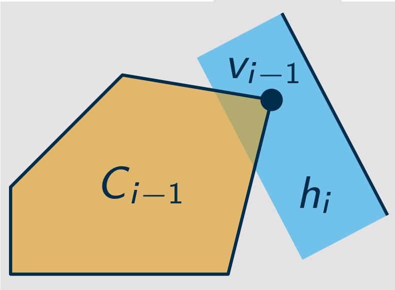
Expected Running Time

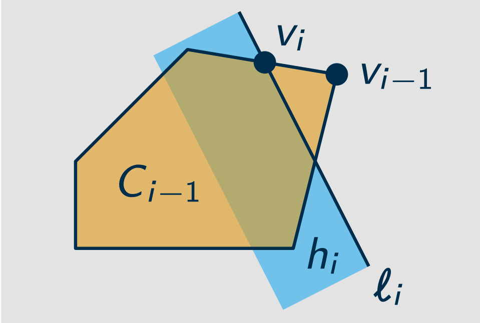
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$



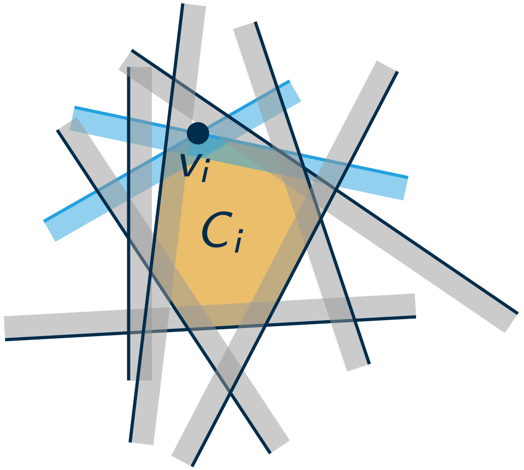


Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i
  - probability: $\frac{2}{i}$
- thus: $P(v_i \neq v_{i-1}) \leq \frac{2}{i}$

Why $\leq$?

if more than two intersect, removing any one might not result in $v_i \neq v_{i-1}$



Randomized Incremental Algorithm

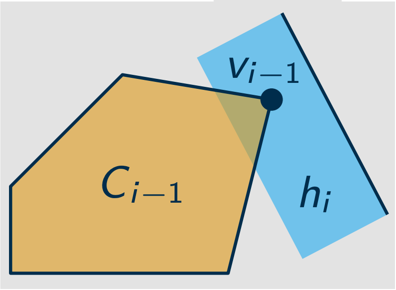
Expected Running Time

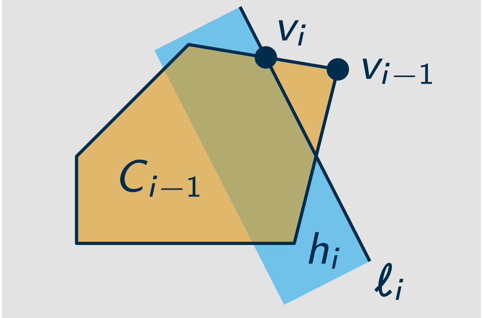
- the running time is a random variable, let's call it X
- additional random variables: X_i is the running time in iteration i

$$X = \sum_{i=1}^{n} X_i \Rightarrow \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

Expected Time In Iteration i:

$$\mathbb{E}[X_i] = O(1) + P(v_{i-1} \notin C_i) \cdot O(i)$$



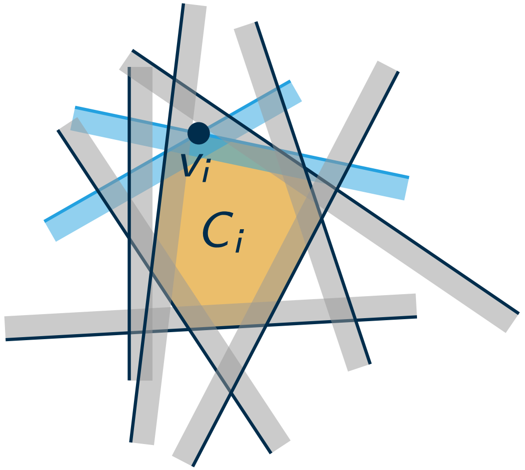


Can We Bound $P(v_{i-1} \notin C_i) = P(v_i \neq v_{i-1})$?

- current view: add random line in each step
- equivalent: draw random line order
- equivalent: remove random line in every step
  - going from C_i to C_{i-1} (removing random h_i)
  - $v_{i-1} \neq v_i \Rightarrow \ell_i$ is one of the two lines that intersect in v_i
  - probability: $\frac{2}{i}$
- thus: $P(v_i \neq v_{i-1}) \leq \frac{2}{i}$

Why $\leq$?

if more than two intersect, removing any one might not result in $v_i \neq v_{i-1}$



$$\Rightarrow \mathbb{E}[X_i] \in O(1) \Rightarrow \mathbb{E}[X] \in O(n)$$

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection
- computing the half-plane intersection: $O(n \log n)$ algorithm

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection
- computing the half-plane intersection: $O(n \log n)$ algorithm
- solving a 2D-LPs: randomized algorithm with expected running time $O(n)$

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection
- computing the half-plane intersection: $O(n \log n)$ algorithm
- solving a 2D-LPs: randomized algorithm with expected running time $O(n)$

What Else Is There?

- the randomized algorithm also works for higher dimensions

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection
- computing the half-plane intersection: $O(n \log n)$ algorithm
- solving a 2D-LPs: randomized algorithm with expected running time $O(n)$

What Else Is There?

- the randomized algorithm also works for higher dimensions
  - running time: still $O(n)$ in expectation, if d constant
  - grows super-exponentially in d

Wrap-Up

What Have We Learned Today?

- computing a 3D mold for casting $\rightarrow$ 2D-LP formulation
- 2D-LP $\rightarrow$ formulate as half-plane intersection
- computing the half-plane intersection: $O(n \log n)$ algorithm
- solving a 2D-LPs: randomized algorithm with expected running time $O(n)$

What Else Is There?

- the randomized algorithm also works for higher dimensions
  - running time: still $O(n)$ in expectation, if d constant
  - grows super-exponentially in d
- this type of randomization (and analysis) works for other geometric problems
  - prominent example: $O(n \log n)$ algo for convex hull in 3D