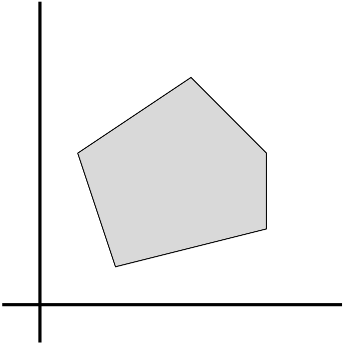
# Integer Programming Methods
# Week 4

Remy Spliet

Erasmus University Rotterdam

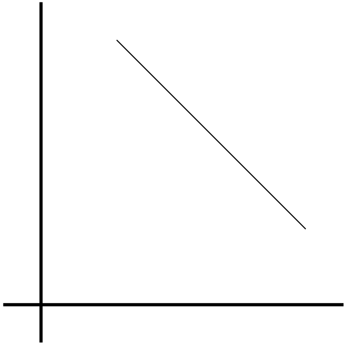
# Dimension of a polyhedron

# Dimension of a polyhedron



- There are two 'independent directions' in this polyhedron.
- Its dimension *should be* 2.

# Dimension of a polyhedron

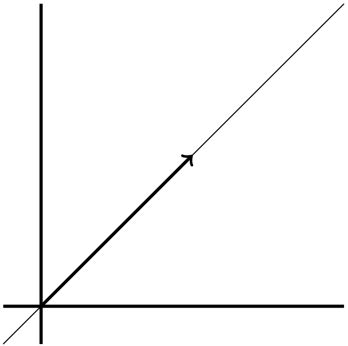


- There is one 'direction' in this polyhedron.
- It's dimension *should be* 1.

# Affine independence

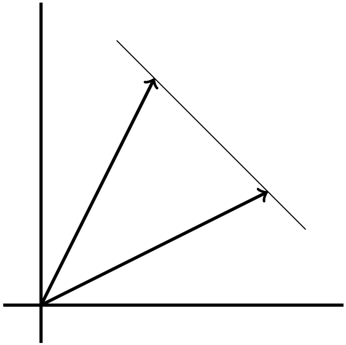
- Recall, for a **subspace**, the dimension is defined as the maximum number of **linear independent** vectors in the subspace.

**Examples:**



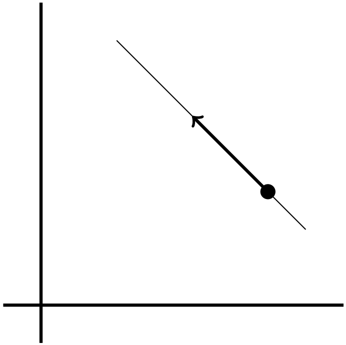
- A line in $\mathbb{R}^2$ has dimension 1.
- A plane in $\mathbb{R}^3$ has dimension 2 (a line in $\mathbb{R}^3$ of course still has dimension 1).

# Affine independence



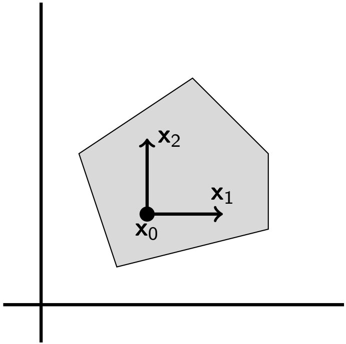
- Two linear independent vectors can be found.
- Still the dimension of this polyhedron should be 1.

# Affine independence



- $\mathbf{0}$ should not be used as a reference point
- Use a different reference point.

# Affine independence




- Use $\mathbf{x}_0$ as a reference point
- The vectors $\mathbf{x}_1 - \mathbf{x}_0$ and $\mathbf{x}_2 - \mathbf{x}_0$ are linearly independent.

# Affine independence

## Definition

The vectors $\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_k$ are **affinely independent** if the vectors $(\mathbf{x}_1 - \mathbf{x}_0)$, ..., $(\mathbf{x}_k - \mathbf{x}_0)$ are linearly independent.

# Affine independence

## Theorem

*The vectors $\mathbf{x}_0$, $\mathbf{x}_1$, …, $\mathbf{x}_k$ are affinely independent if and only if*

$$\left.\begin{array}{r} \sum_{i=0}^{k} c_i = 0 \\ \sum_{i=0}^{k} c_i \mathbf{x}_i = \mathbf{0} \end{array}\right\} \Leftrightarrow \; c_i = 0, \; i = 0, \ldots, k$$

**Or equivalently:**

The vectors $\mathbf{x}_0$, $\mathbf{x}_1$, …, $\mathbf{x}_k$ are affinely independent if and only if $\begin{bmatrix} 1 \\ \mathbf{x}_0 \end{bmatrix}, \begin{bmatrix} 1 \\ \mathbf{x}_1 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ \mathbf{x}_k \end{bmatrix}$ are linearly independent.

# Dimension

## Definition

If the maximum number of affinely independent vectors in a polyhedron $P$ is $k$ then $P$ is said to be of **dimension** $k - 1$, denoted $\dim(P) = k - 1$.

# Properties

## Theorem

*For the polyhedron $P \subseteq \mathbb{R}^n$ it holds that*

$$\dim(P) \leq n$$

*If $\dim(P) = n$ it is said to be* **full dimensional***.*

## Proof.

- The maximum number of linearly independent vectors in $\mathbb{R}^n$ is $n$.
- Therefore, the maximum number of affinely independent vectors in $\mathbb{R}^n$ is $n + 1$.

□

# Properties

## Theorem

*Consider the non-empty polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} = \mathbf{b}, D\mathbf{x} \leq \mathbf{d}\}$, such that there exists an $\mathbf{x} \in P$ for which $D\mathbf{x} < \mathbf{d}$. The dimension of $P$ is given by*

$$\dim(P) = n - rank([A\ \mathbf{b}])$$

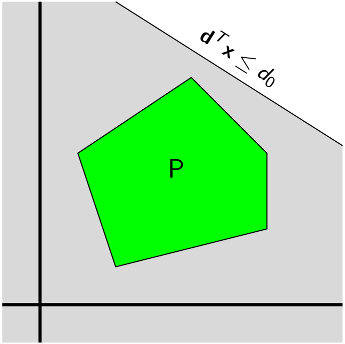
## Corollary

*Consider a non-empty polyhedron $P$. For any matrix $A$ and vector $\mathbf{b}$ such that $A\mathbf{x} = \mathbf{b}$ for all $\mathbf{x} \in P$ it holds that*

$$\dim(P) \leq n - rank([A\ \mathbf{b}])$$

# Facets

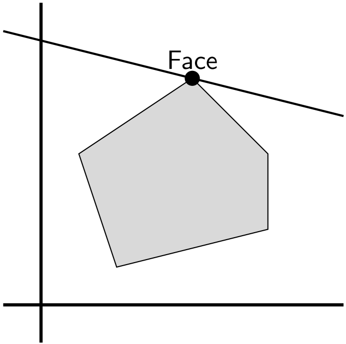
# Valid inequalities



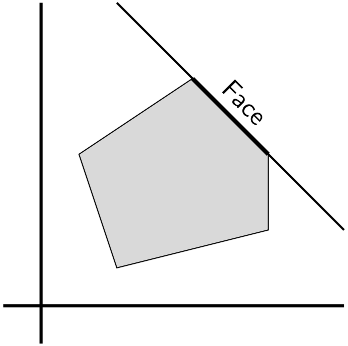

## Definition

Consider the polyhedron $P \subseteq \mathbb{R}^n$, the vector $\mathbf{d} \in \mathbb{R}^n$ and scalar $d_0$. If the inequality $\mathbf{d}^T\mathbf{x} \leq d_0$ is satisfied by all $\mathbf{x} \in P$, it is called a **valid inequality** for $P$.

# Faces



# Faces



# Faces





## Definition

Consider the polyhedron $P$ and valid inequality $\mathbf{d}^T\mathbf{x} \leq d_0$ and define

$$F = \{x \in P | \mathbf{d}^T\mathbf{x} = d_0\}$$

The set $F$ is called a **face** of $P$.

Furthermore, $F$ it is called a **proper face** if $F \neq \emptyset$ and $F \neq P$.

**Note:**

- Observe that a face is a **polyhedron**.

# Facets





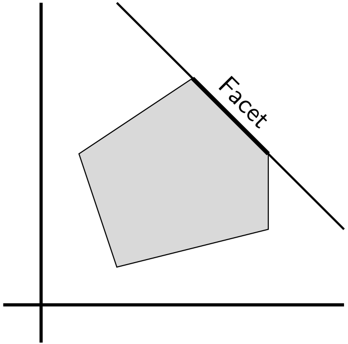
## Definition

An inclusionwise maximal proper face of a polyhedron is called a **facet**. In this case the valid inequality corresponding to $F$ is said to be **facet defining**.

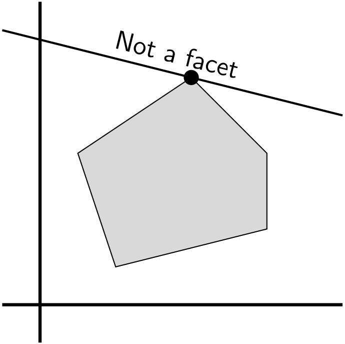
## Theorem

*A face $F$ of the polyhedron $P$ is a* **facet** *if and only if $F$ is nonempty and* $\dim(F) = \dim(P) - 1$.

# Facets



# Facets



# Minimal representation





✓ Of **every facet** of a polyhedron $P$, exactly one facet defining inequality is **necessary** to represent $P$.

✗ All **faces** $F$ with $\dim(F) < \dim(P) - 1$ are **redundant** to represent the polyhedron $P$.

# Formulations

# Formulations

- Integer feasible region:

$$X = \{\mathbf{x} \in \mathbb{N}^n | A\mathbf{x} \leq \mathbf{b}\}$$

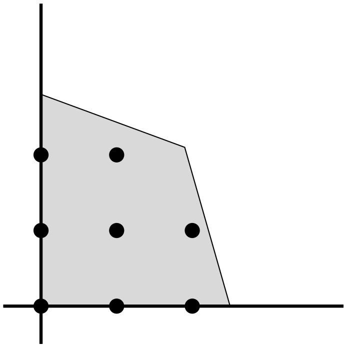
- ILP:

$$\max\{\mathbf{c}^T\mathbf{x} | \mathbf{x} \in X\}$$

# Formulations

## Definition

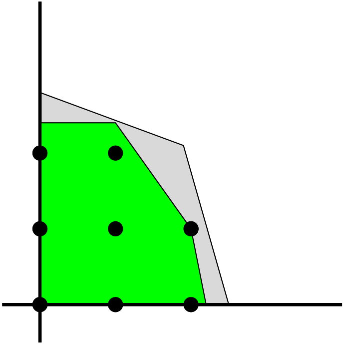
The (polyhedral) set $P \subseteq \mathbb{R}^n$ is a **formulation** of $X \subseteq \mathbb{N}^n$ if and only if $X = P \bigcap \mathbb{N}^n$. The LP relaxation using formulation $P$ is $\max\{\mathbf{c}^T\mathbf{x} | \mathbf{x} \in P\}$



# Formulations

## Definition

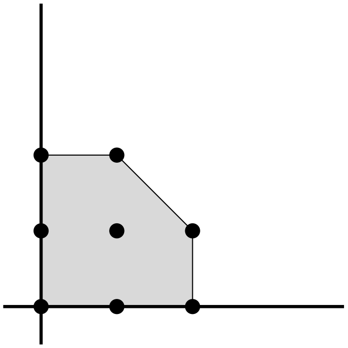
Consider the formulations $P_1$ and $P_2$ of $X$. Formulation $P_1$ is **stronger** than $P_2$ if $P_1 \subset P_2$.



# Formulations

## Theorem

*The LP problem* $\max\{\mathbf{c}^T\mathbf{x} | \mathbf{x} \in conv(X)\}$ *gives the optimal integer solution to* $\max\{\mathbf{c}^T\mathbf{x} | \mathbf{x} \in X\}$ *and hence the* **strongest** *formulation of* $X \subseteq \mathbb{N}^n$ *is the polyhedron* $conv(X)$*.*



# Formulations





- **x** We usually do not know inequalities describing $\text{conv}(X)$.
- **?** How to find facet defining inequalities of $\text{conv}(X)$?
- **?** Should we find (all) facet defining inequalities of $\text{conv}(X)$?

# Extended formulations

- We discussed adding or removing **inequalities**.

✓ A change of **variables** might also be helpful.

! The concept of **projection** plays a role in this.

# Extended formulations

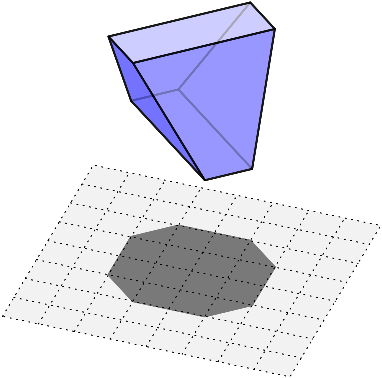




## Definition

The orthogonal **projection** of a set $S \subset \mathbb{R}^{n+p}$ onto $\mathbb{R}^n$ is

$$\text{proj}_x(S) = \{\mathbf{x} \in \mathbb{R}^n : \exists \mathbf{z} \in \mathbb{R}^p, (\mathbf{x}, \mathbf{z}) \in S\}$$



# Extended formulations

### Definition

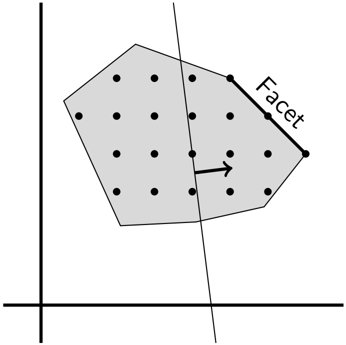
A set of linear inequalities describing $S$, is called an **extended formulation** of $\text{proj}_x(S)$.

- ✗ An extended formulation might have more variables.
- ✓ An extended formulation might have fewer facets.

# Branch-and-cut

# Branch-and-cut

- Some (facet defining) inequalities are more helpful than others.



# Branch-and-cut

- x More inequalities $\Rightarrow$ slower simplex algorithm.
- ✓ More inequalities $\Rightarrow$ better bounds.
- Only add a few valid inequalities, which improve the bound the most.
- **Relax** valid inequalities **and add** them when violated.
- **Branch**, when no more inequalities are added.

# Row generation

- A **cutting plane** is a valid inequality that is violated by the current solution to the LP relaxation.
- Iteratively adding cutting planes is called **row generation**.
- Identifying a cutting plane is called **separation**.

# Row generation



**Different perspectives on row generation:**

- Let $P_1$ be a formulation of $X$.
- Let $A\mathbf{x} \leq \mathbf{b}$ be valid inequalities for $X$.
- There exist an $\mathbf{x} \in P_1$, and row $i$, such that $A_i\mathbf{x} > \mathbf{b}_i$.
- So $P_2 = \{\mathbf{x} \in P_1 | A\mathbf{x} \leq \mathbf{b}\}$ is a stronger formulation for $X$.

Perspective 1 Use $P_1$ and **strengthen** by adding violated inequalities from $A\mathbf{x} \leq \mathbf{b}$.

Perspective 2 Use $P_2$ and **relax** some of its constraints (namely $A\mathbf{x} \leq \mathbf{b}$) and only add them when violated.

! *Actually, $P_1$ need not be a formulation of $X$, as long as $P_2$ is.*

# Row generation



**Separation algorithm for $A\mathbf{x} \leq \mathbf{b}$:**

Input:
- A solution $\mathbf{x}^*$ to an LP-relaxation.

Output:
- One or multiple valid inequalities from $A\mathbf{x} \leq \mathbf{b}$ that are violated by $\mathbf{x}^*$.

  OR

- A guarantee that $A\mathbf{x}^* \leq \mathbf{b}$.

# Row generation

The **separation problem** is the problem of finding the valid inequality $\mathbf{a}_i \mathbf{x}^* \leq \mathbf{b}_i$, for $i = 1, \ldots m$ that is violated the most:

$$\arg\max_{i=1,\ldots,m} \{\mathbf{a}_i^T \mathbf{x}^* - \mathbf{b}_i\}$$

# Row generation

- ✗ The separation problem might be **difficult**.
- ✗ The separation algorithm is usually **run very often**.
- ✓ **Heuristics** can be used to solve the separation problem.

# Summary of branch-and-cut

Step 1 **Initialize** the branching tree with a single node corresponding to the LP relaxation of the ILP.

Step 2 **Select** a node from the tree.

Step 3 **Solve** the LP of the current node.

Step 4 **Separate** valid inequalities.
- If found, add and return to Step 3.

Step 5 **Branch and/or prune**.
- If the optimal ILP solution is not found return to Step 2.

**Note:** If in step 3 column generation is used, the algorithm is called **branch-price-and-cut**.

# Computational considerations

- ✓ More inequalities $\Rightarrow$ smaller LP gap $\Rightarrow$ less nodes.
- x More inequalities $\Rightarrow$ slower simplex algorithm.
- x More separated inequalities $\Rightarrow$ more separation time.

**There is a trade-off between the time spent per node, and the total number of nodes processed.**

# Computational considerations

**Limiting the number of inequalities:**

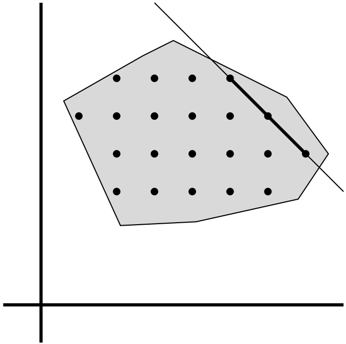
- Put a **limit** on the total number of inequalities that are separated.
- Only add inequalities that are violated by more than some **threshold**.
- **Remove** unimportant inequalities.
  - E.g., keep track of which inequalities were not binding in the last couple of iterations.

# Computational considerations

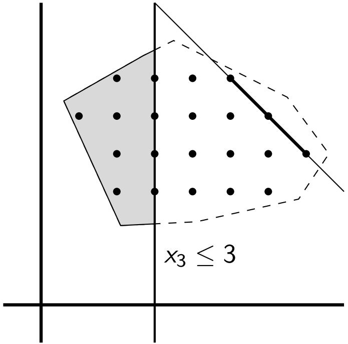
**Different inequalities per node:**

- ✓ A valid inequality is also feasible in every branching node.
- ✗ A valid inequality might not help in every branching node.

# Computational considerations



# Computational considerations



# Computational considerations





**Different inequalities per node:**

- ★ Remove the old inequalities for each new node.
  - We **risk** generating the same inequalities very often.
  - Store removed inequalities in a **pool**.
  - **Check the pool** for violated inequalities before running the separation algorithm.
- ★ Separate **node specific** inequalities.