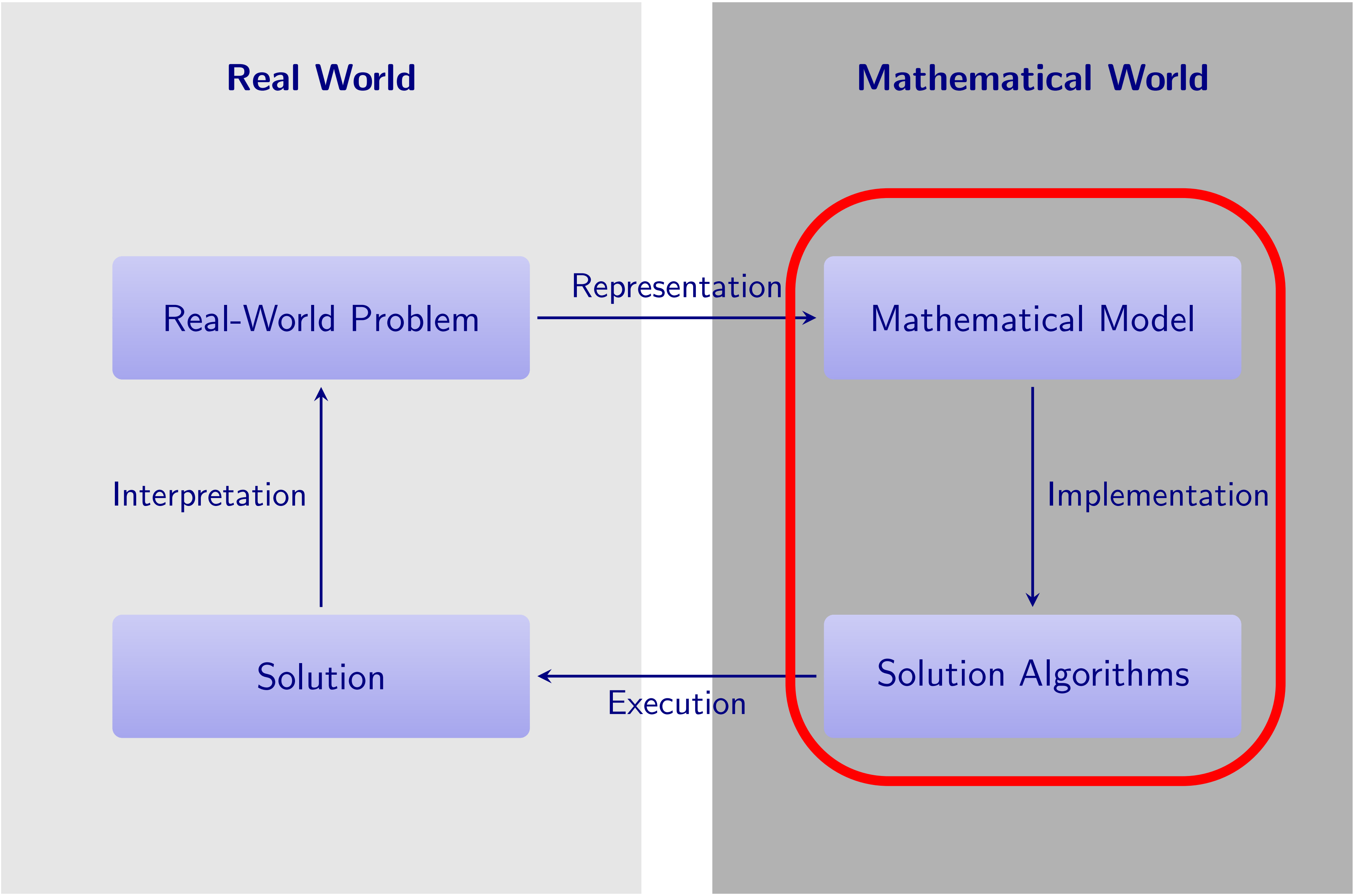
Mathematical Tools for Engineering and Management

Lecture 3

2 Nov 2011


Real World
Mathematical World
Real-World Problem
Representation
Mathematical Model
Interpretation
Implementation
Solution
Execution
Solution Algorithms

$$\text{maximize/minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for all } i = 1, \ldots, m$$

$$\ell_j \le x_j \le u_j \quad \text{for all } j = 1, \ldots, n$$

General form of LPs

$$\text{maximize/minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for all } i = 1, \ldots, m$$

$$\ell_j \leq x_j \leq u_j \quad \text{for all } j = 1, \ldots, n$$

General form of LPs

➡ Every LP can be written in the following standard form:

Standard form of LPs

$$\text{maximize/minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for all } i = 1, \ldots, m$$

$$\ell_j \le x_j \le u_j \quad \text{for all } j = 1, \ldots, n$$

General form of LPs

➡ Every LP can be written in the following standard form:

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for all } i = 1, \ldots, m$$

maximize/minimize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for all $i = 1, \ldots, m$

$\ell_j \leq x_j \leq u_j$ for all $j = 1, \ldots, n$

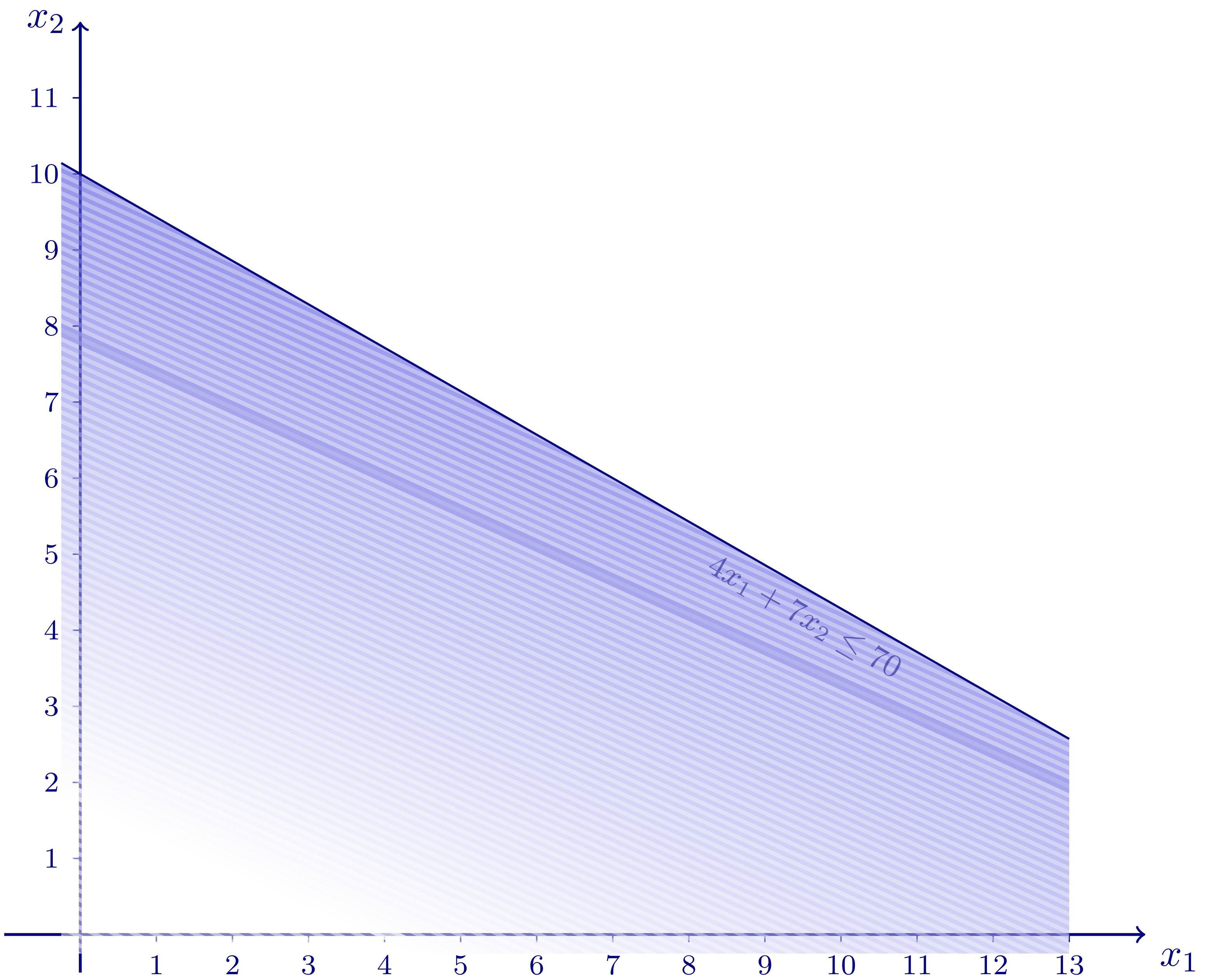
General form of LPs

➡ Every LP can be written in the following standard form:

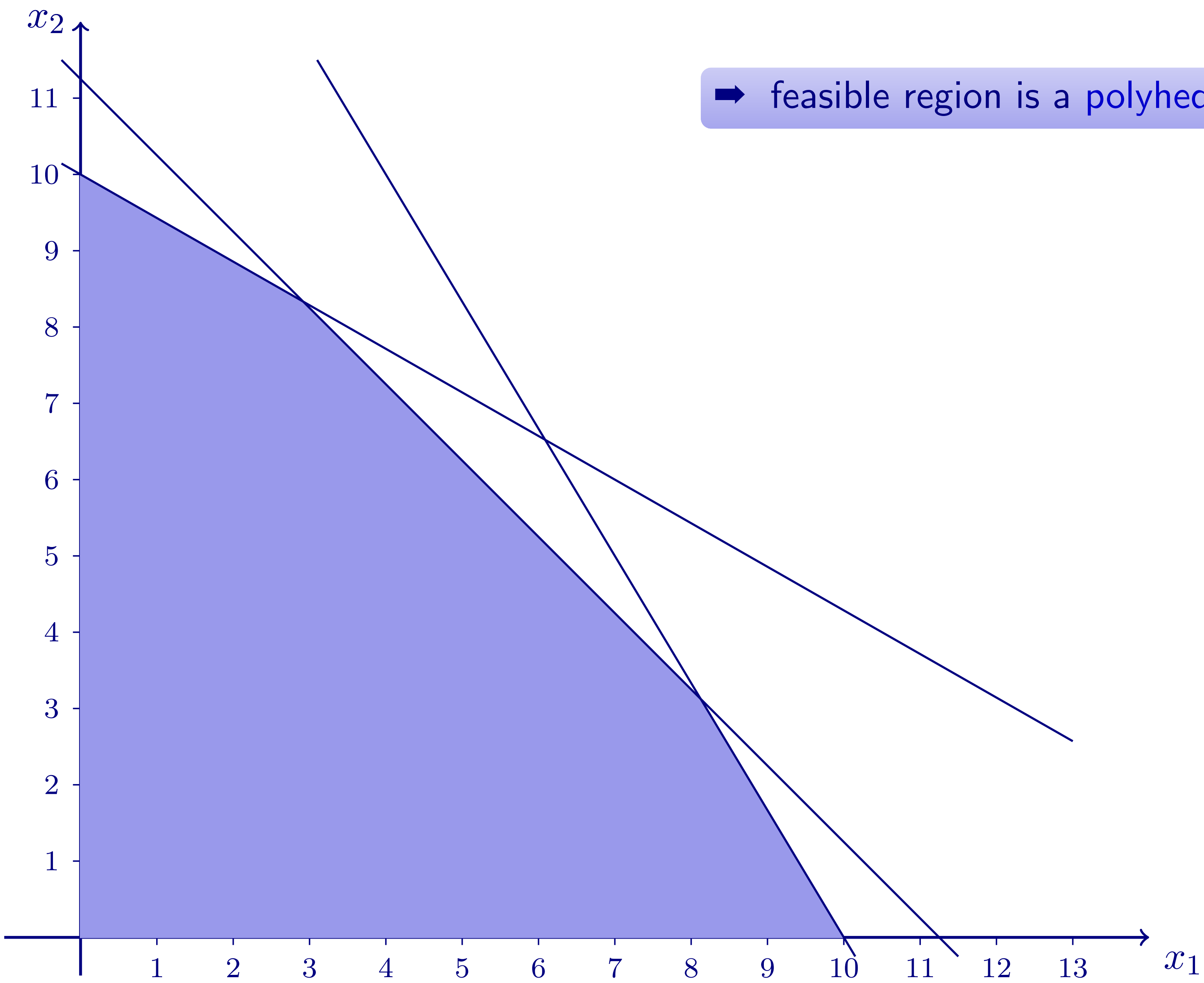
maximize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for all $i = 1, \ldots, m$

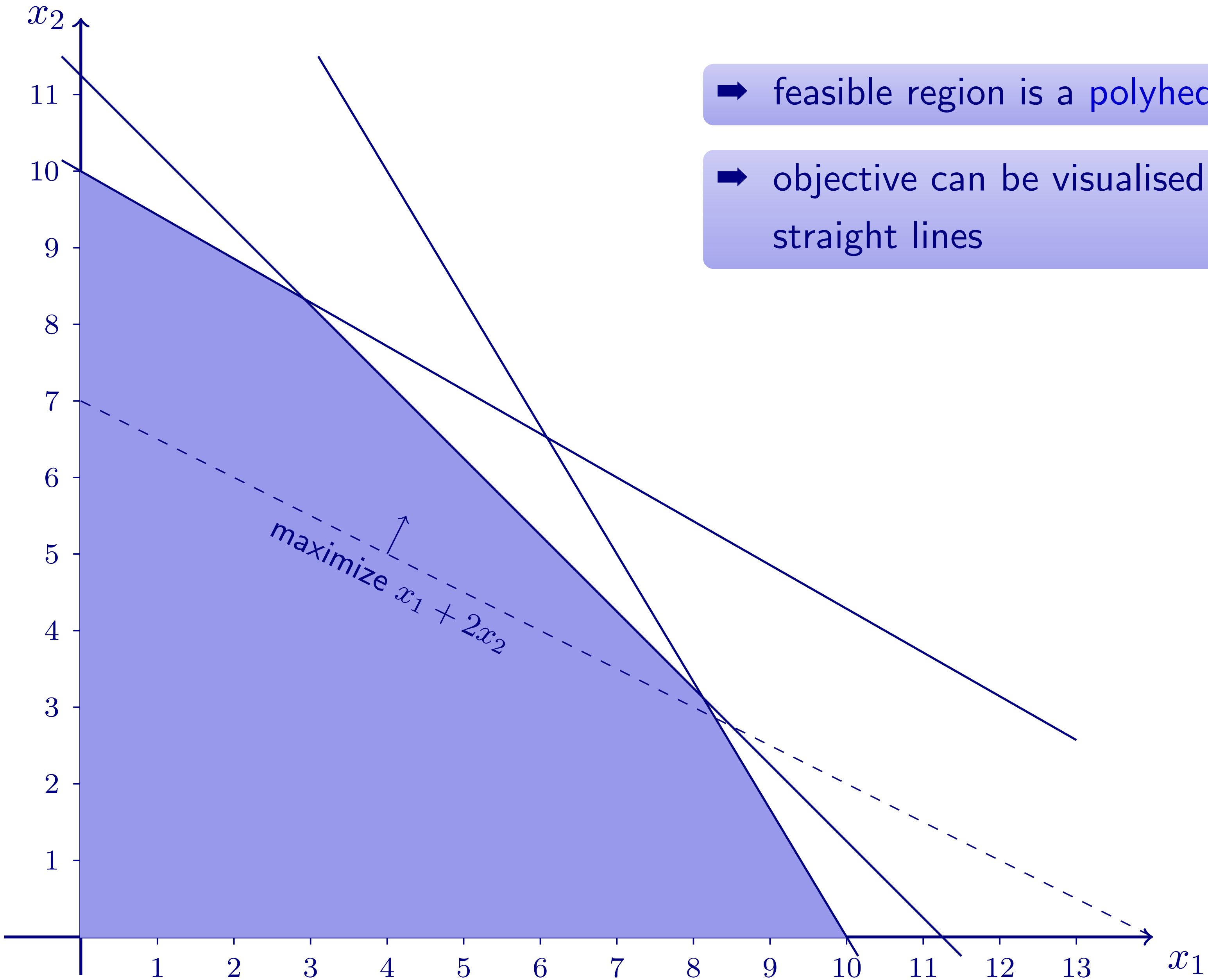
➡ NOTE:
- Every LHS and the objective are linear functions!
- Every constraint is a $\leq$-constraint!



x_2
11
10
9
8
7
6
5
4
3
2
1
$4x_1 + 7x_2 \leq 70$
1
2
3
4
5
6
7
8
9
10
11
12
13
x_1

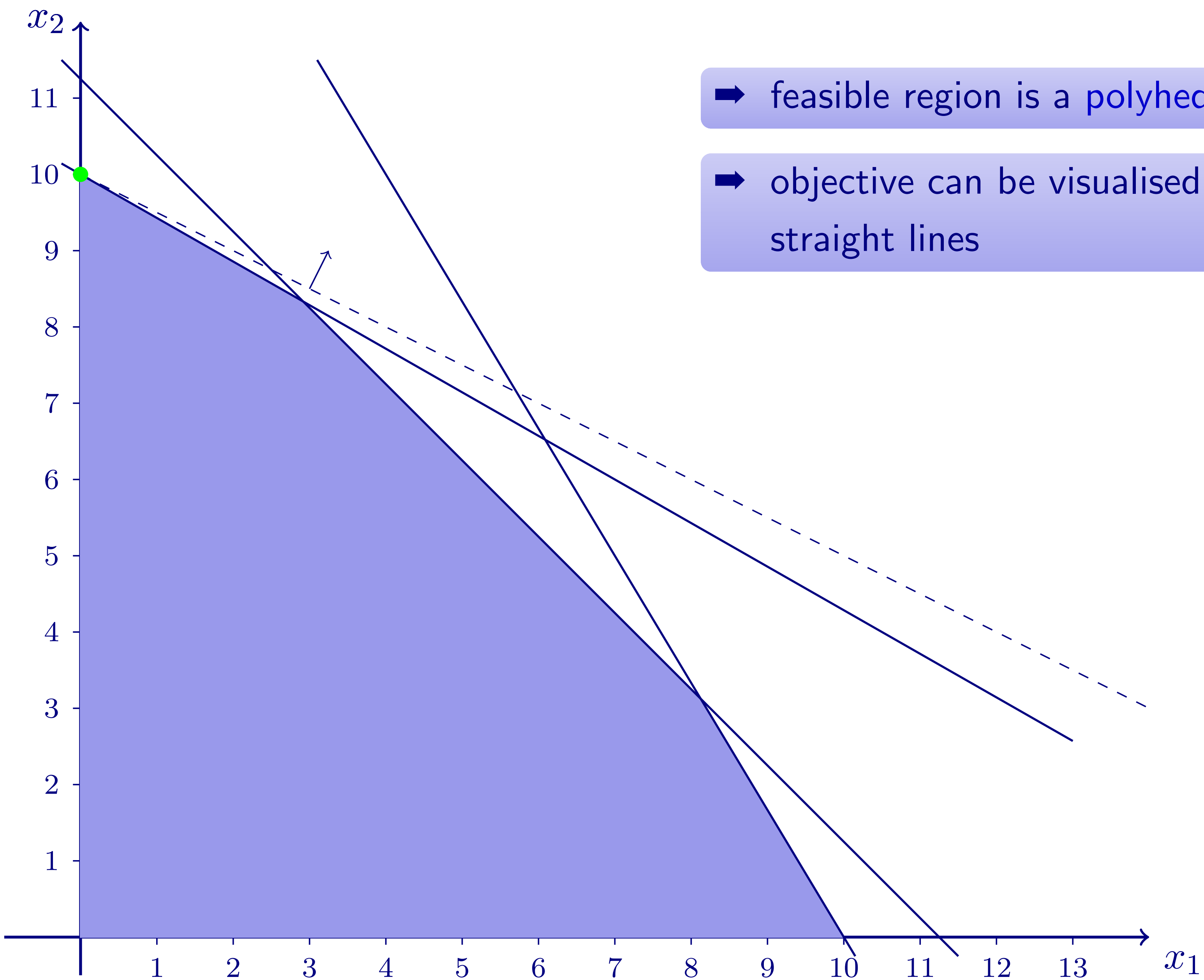


➡ feasible region is a polyhedron

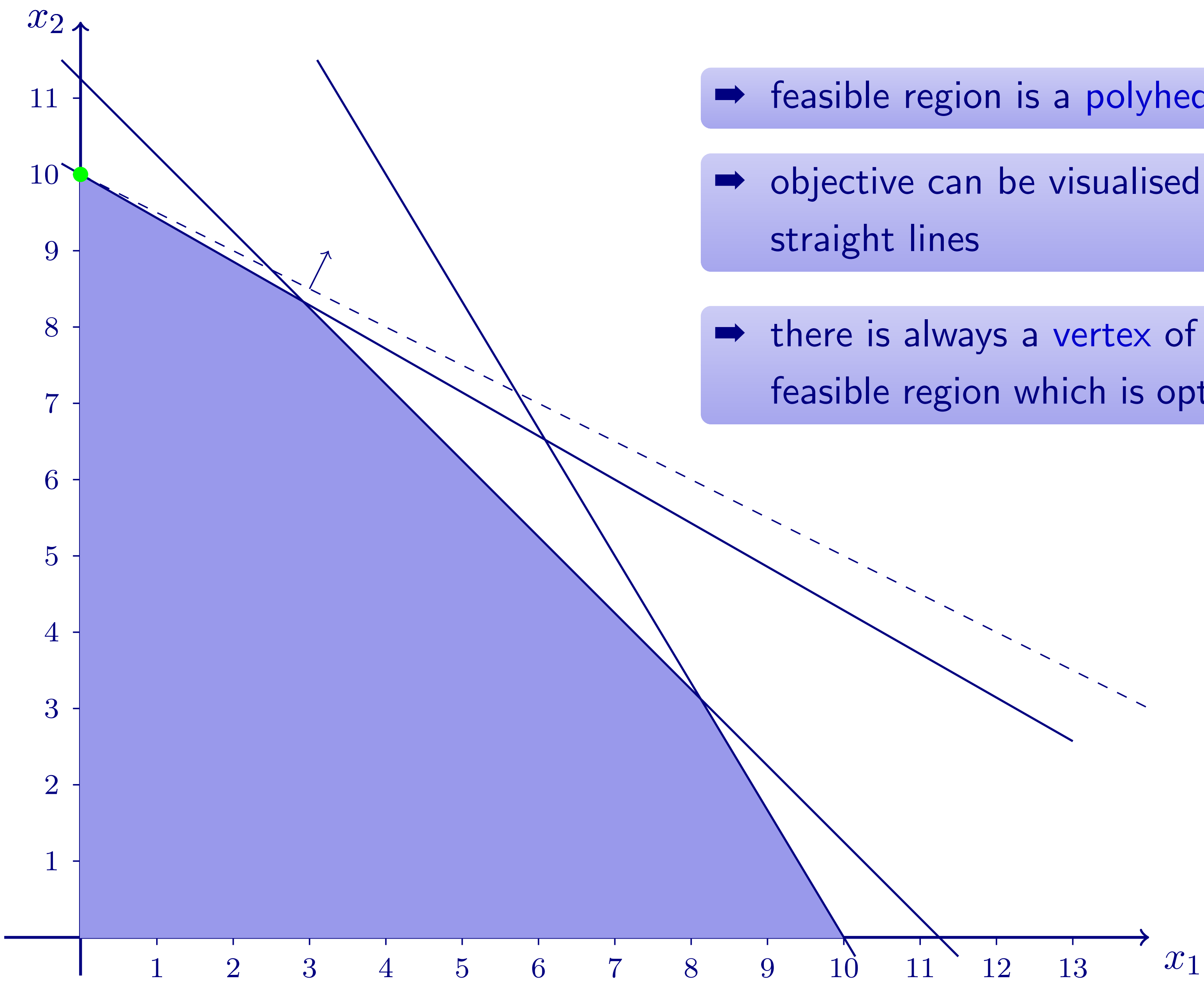


➡ feasible region is a polyhedron

➡ objective can be visualised by straight lines

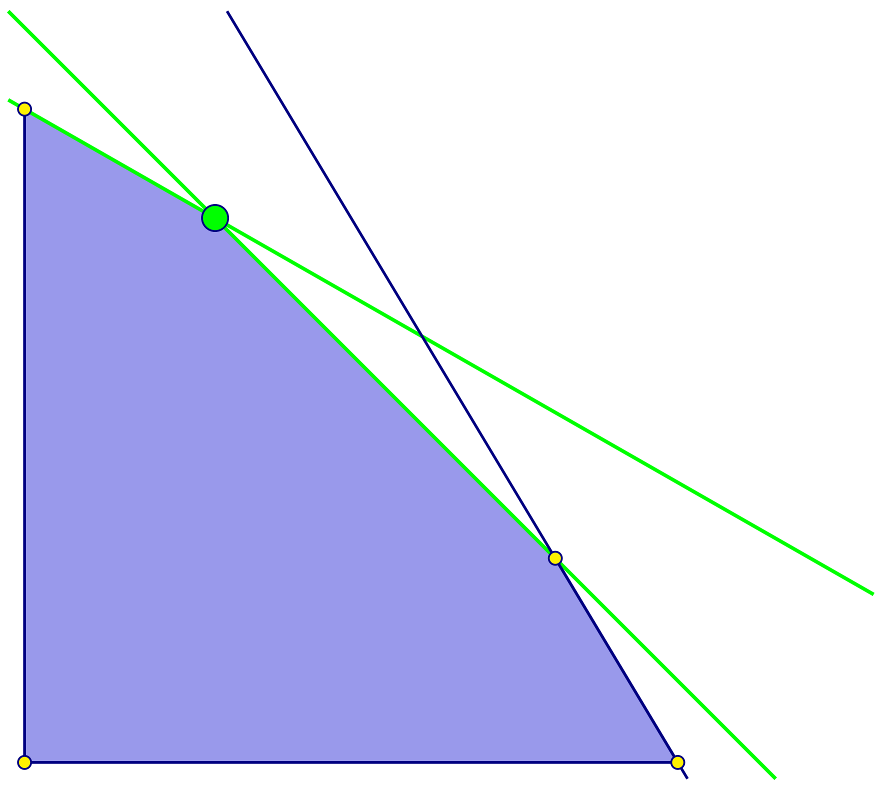


- ➡ feasible region is a polyhedron
- ➡ objective can be visualised by straight lines

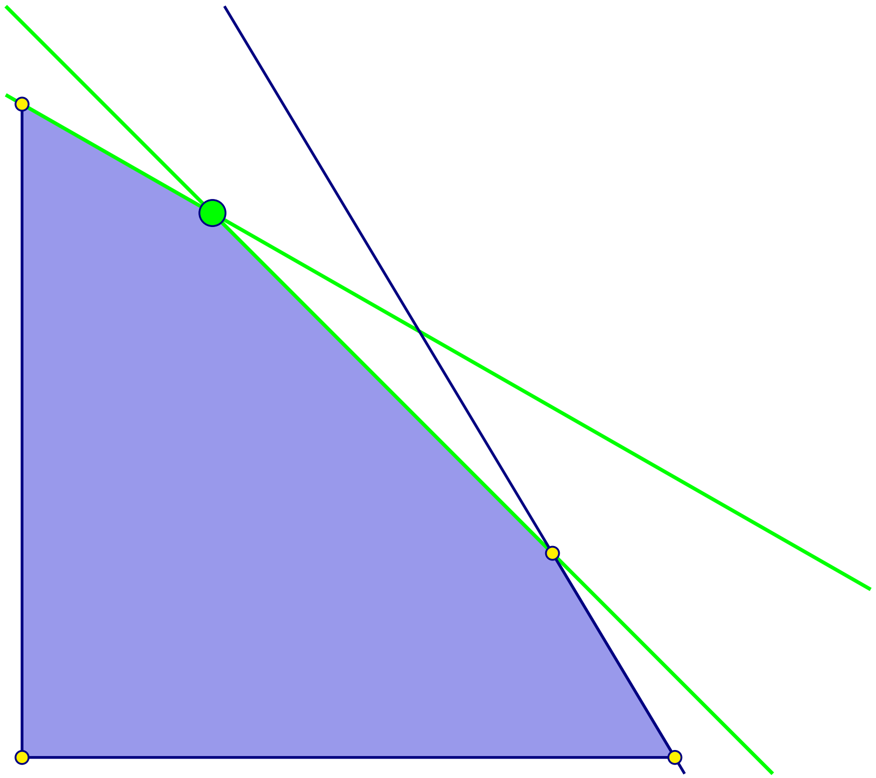


- feasible region is a polyhedron
- objective can be visualised by straight lines
- there is always a vertex of the feasible region which is optimal

- LP with 2 variables

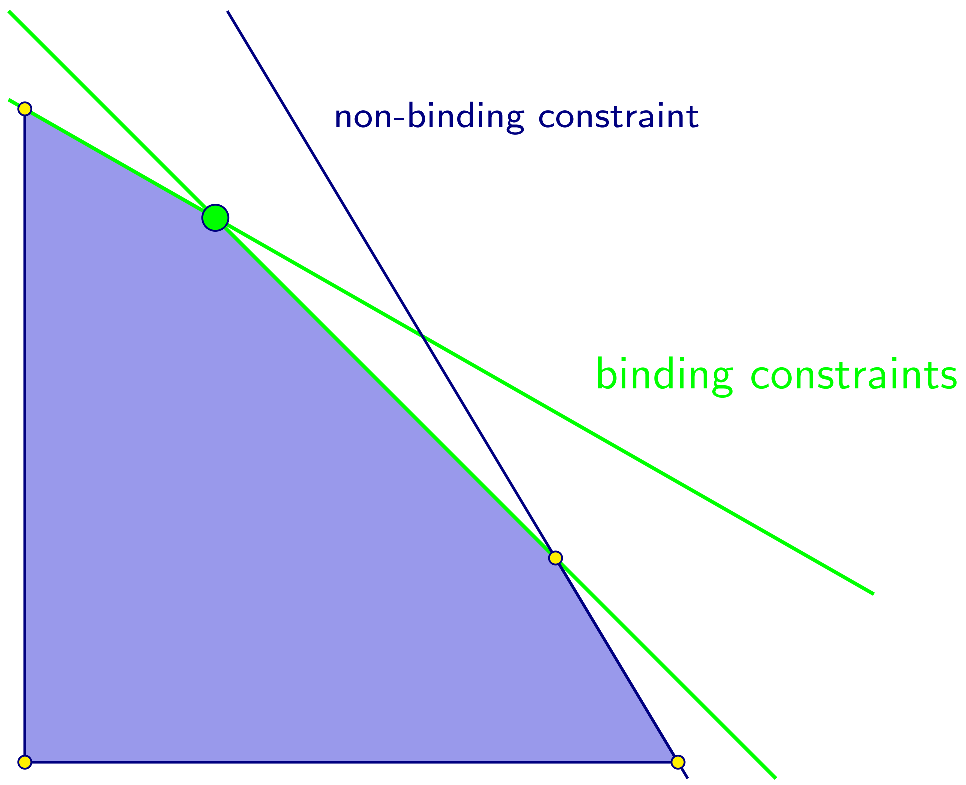


- LP with 2 variables
  - ➡ Vertex is the intersection of 2 lines, given by 2 binding constraints



▷ LP with 2 variables

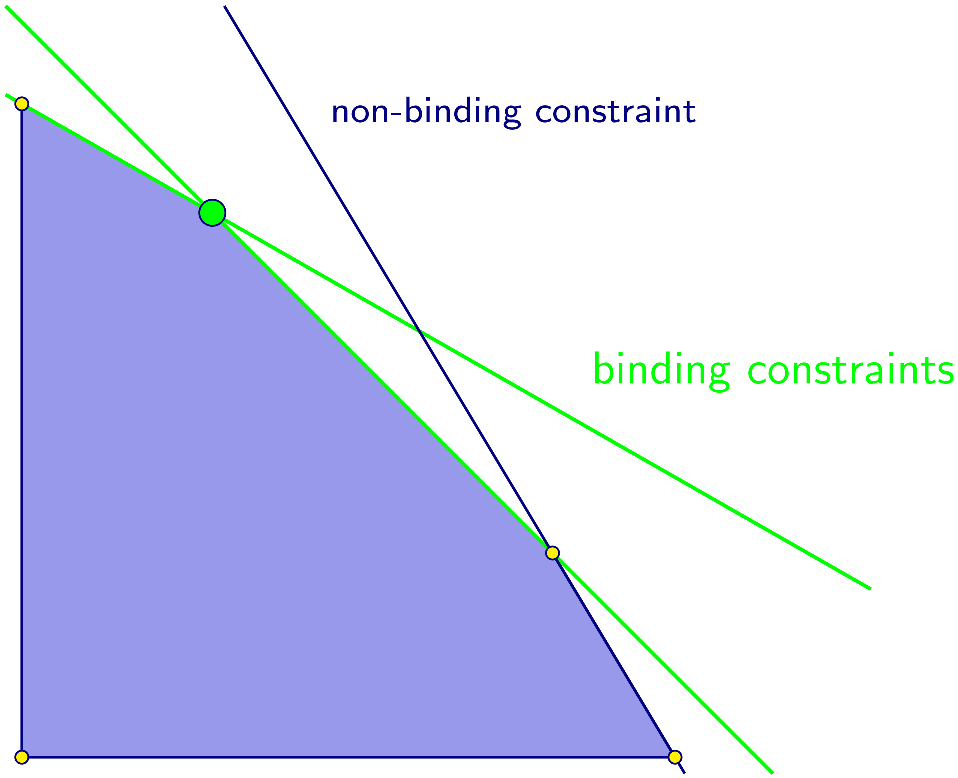
➡ Vertex is the intersection of 2 lines, given by 2 binding constraints



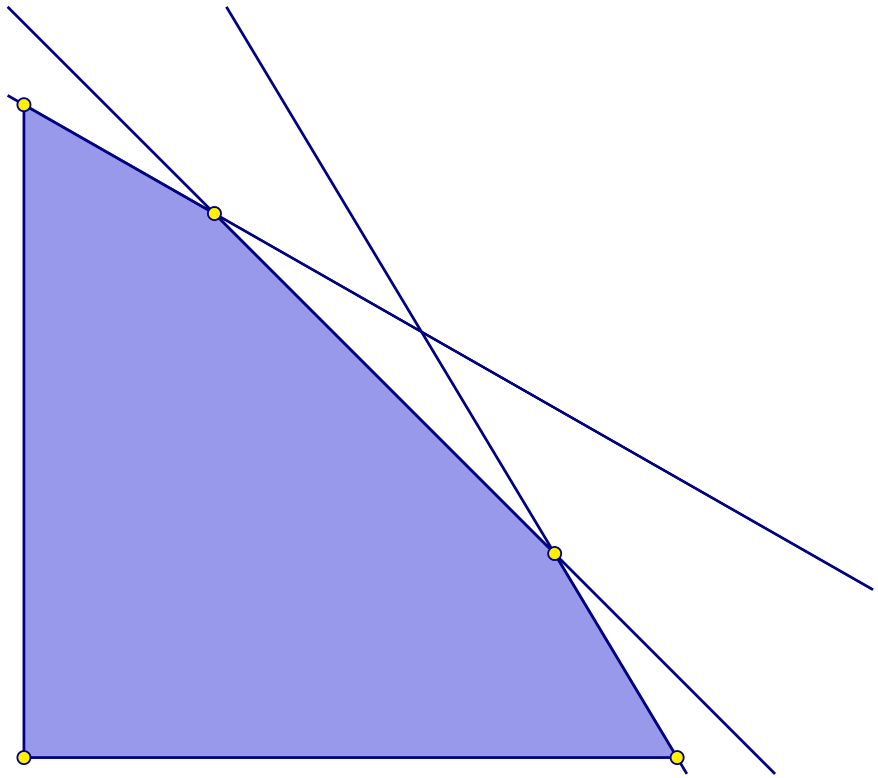
▷ LP with 2 variables

➡ Vertex is the intersection of 2 lines, given by 2 binding constraints

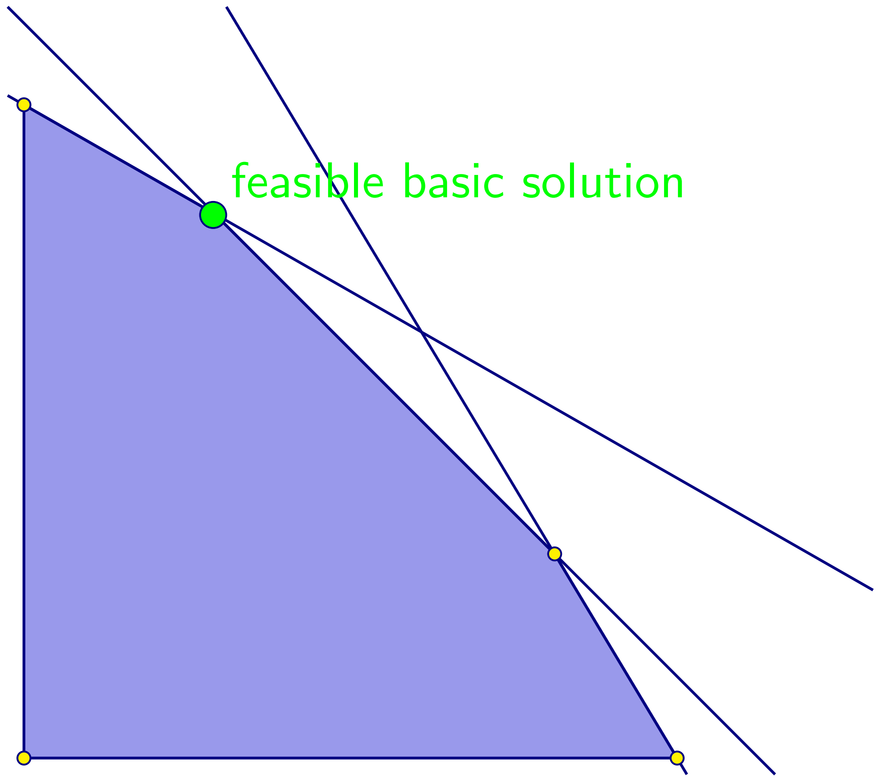
➡ Compute its coordinates by solving a system of 2 linear equations in 2 variables



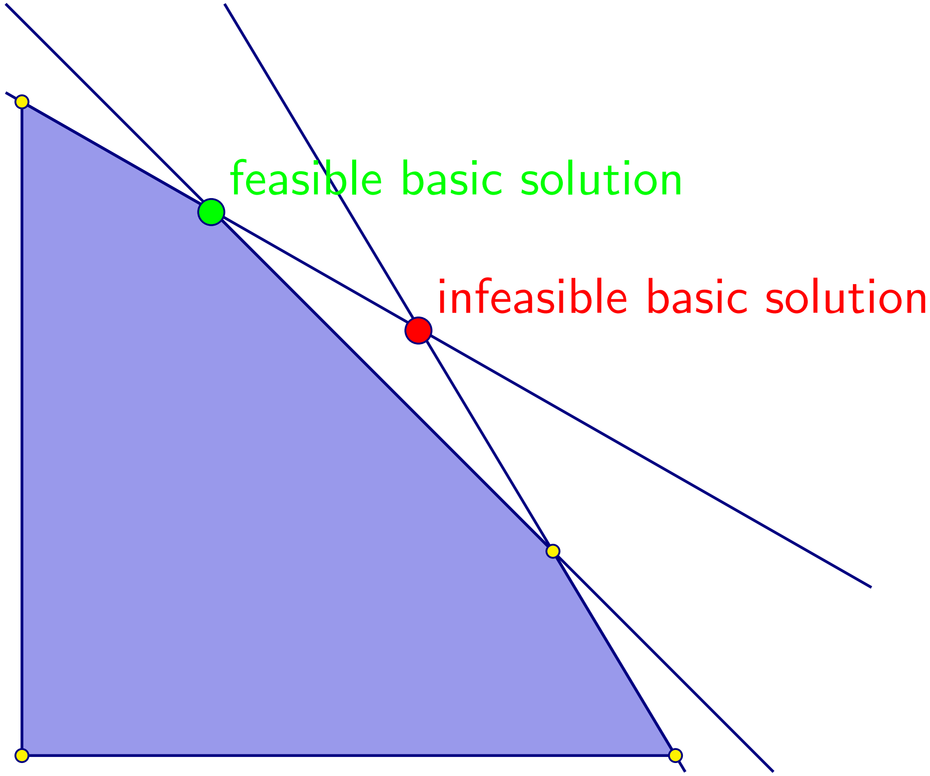
▷ Solutions of a system of 2 linear equations given by constraints are called basic solutions



- Solutions of a system of 2 linear equations given by constraints are called **basic solutions**

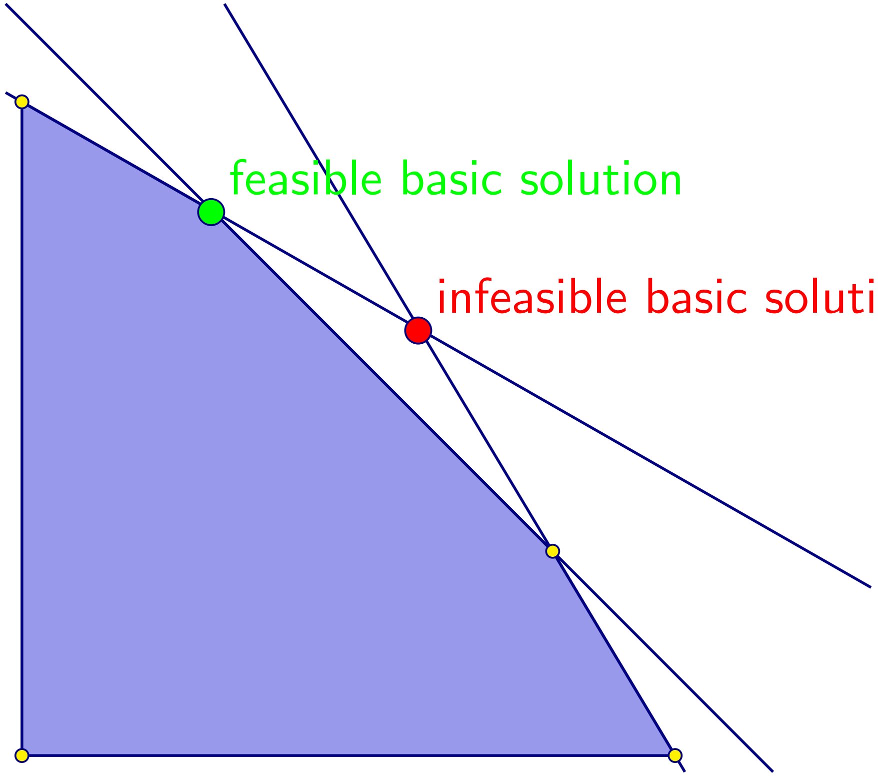


- Solutions of a system of 2 linear equations given by constraints are called **basic solutions**



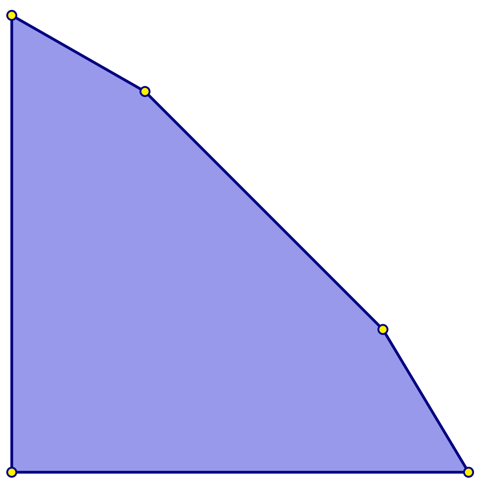
- Solutions of a system of 2 linear equations given by constraints are called **basic solutions**

- **Feasible basic solutions** are exactly the vertices of the feasible region

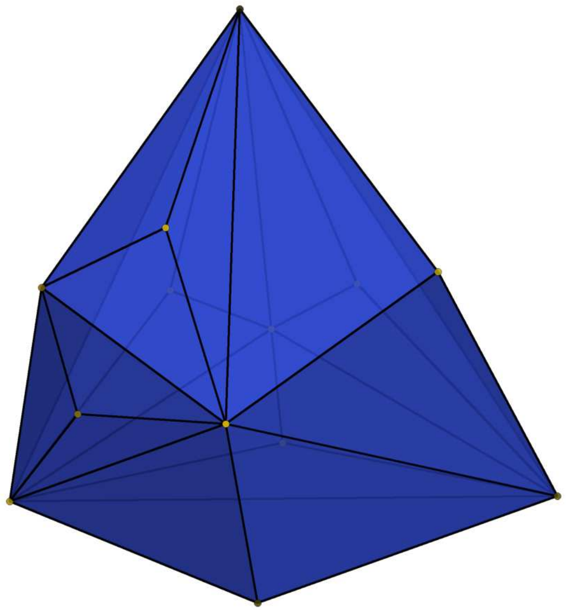


▷ LP with n variables

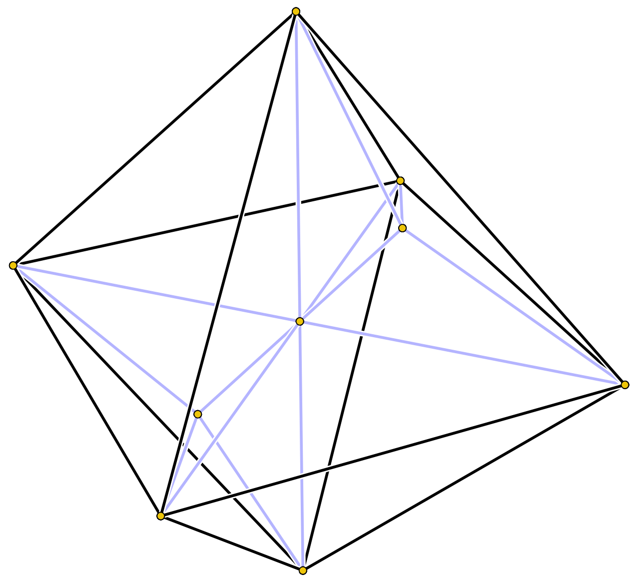
2-dim



3-dim

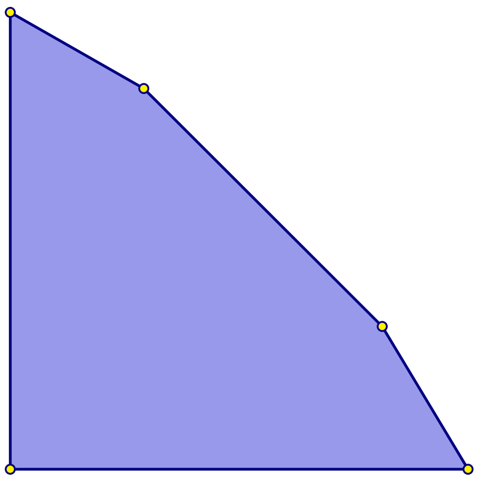


4-dim

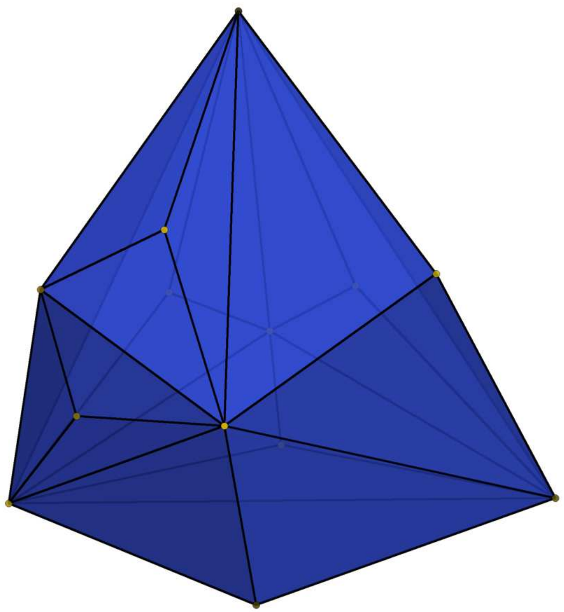


- ▷ LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints

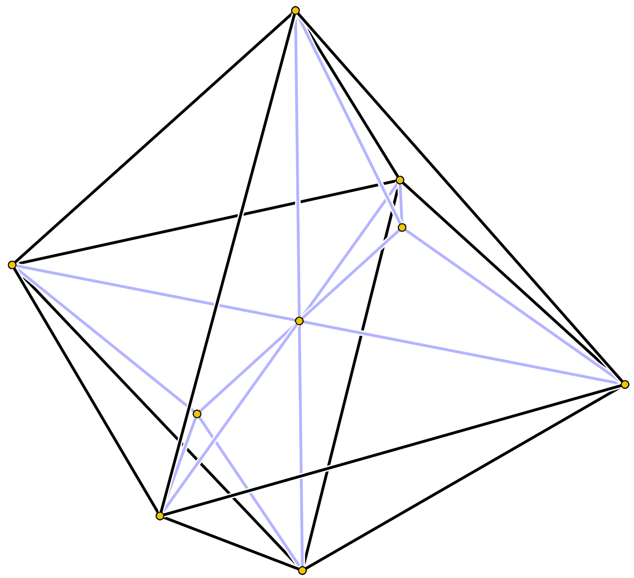
2-dim



3-dim

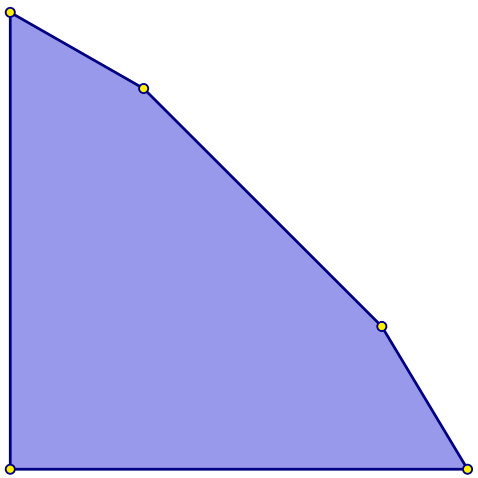


4-dim

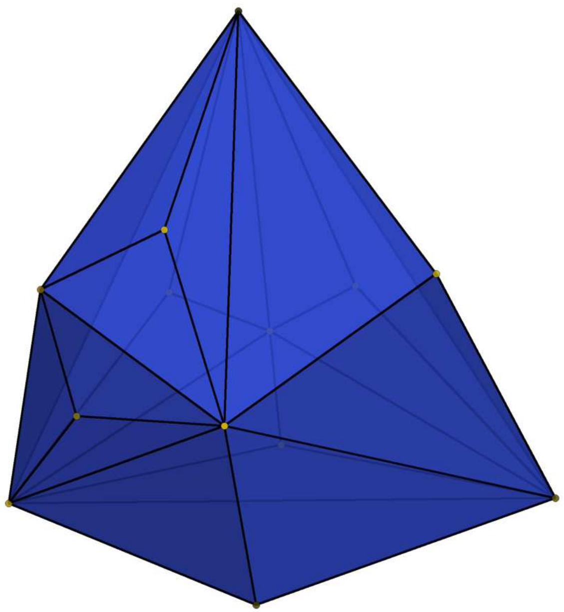


- LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *

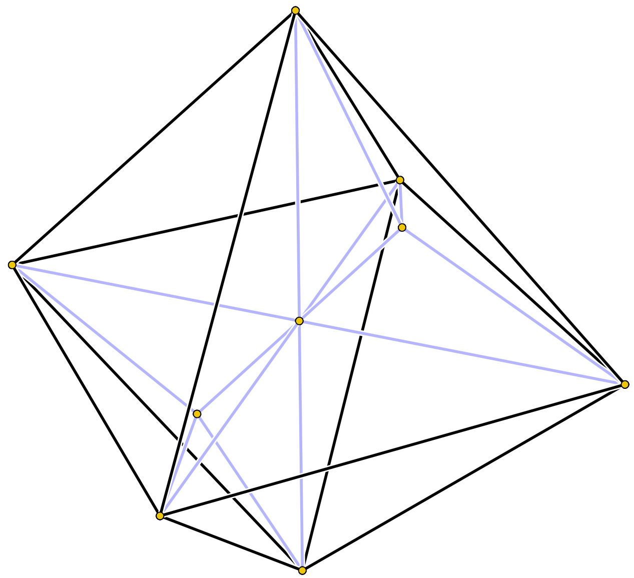
2-dim



3-dim



4-dim



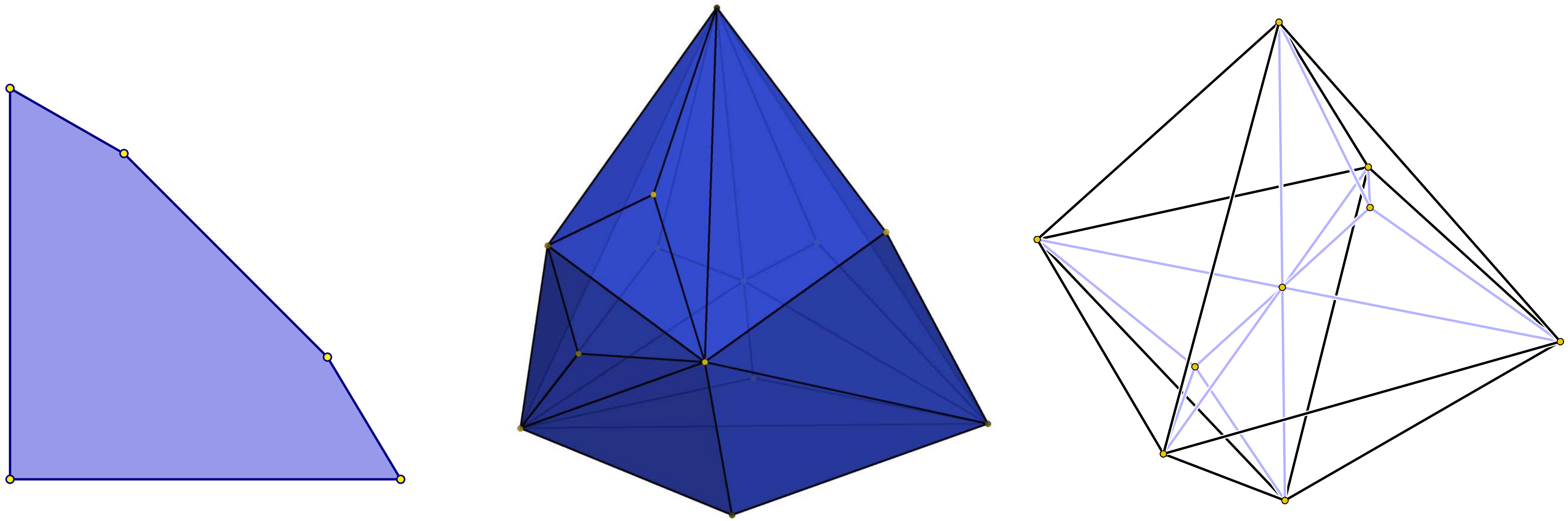
* constraints have to be linearly independent!

▷ LP with n variables

➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *

➡ Compute its coordinates by solving a system of n linear equations in n variables

2-dim 3-dim 4-dim



* constraints have to be linearly independent!

- ▷ LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- ▷ Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)

* constraints have to be linearly independent!

- ▷ LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- ▷ Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)
  - There is always a vertex which is optimal (if there is an optimum at all)

* constraints have to be linearly independent!

▷ LP with n variables

➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *

➡ Compute its coordinates by solving a system of n linear equations in n variables

▷ Still true in n dimensions:

- The feasible region is a polyhedron (possibly empty)
- There is always a vertex which is optimal (if there is an optimum at all)

▷ Idea of the Simplex Algorithm: Jump from vertex to vertex in the direction of the objective vector until an optimal vertex is reached

* constraints have to be linearly independent!

- LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)
  - There is always a vertex which is optimal (if there is an optimum at all)
- Idea of the Simplex Algorithm: Jump from vertex to vertex in the direction of the objective vector until an optimal vertex is reached
  - ➡ Details:

How to get to a neighbouring vertex?

* constraints have to be linearly independent!

- ▷ LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- ▷ Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)
  - There is always a vertex which is optimal (if there is an optimum at all)
- ▷ Idea of the Simplex Algorithm: Jump from vertex to vertex in the direction of the objective vector until an optimal vertex is reached
  - ➡ Details:

How to get to a neighbouring vertex?

How to realise that the optimum is reached?

* constraints have to be linearly independent!

- LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)
  - There is always a vertex which is optimal (if there is an optimum at all)
- Idea of the Simplex Algorithm: Jump from vertex to vertex in the direction of the objective vector until an optimal vertex is reached
  - ➡ Details:

How to get to a neighbouring vertex?

How to realise that the optimum is reached?

What if there is no optimum?

* constraints have to be linearly independent!

- ▷ LP with n variables
  - ➡ Vertex is the intersection of n hyperplanes, given by n binding constraints *
  - ➡ Compute its coordinates by solving a system of n linear equations in n variables
- ▷ Still true in n dimensions:
  - The feasible region is a polyhedron (possibly empty)
  - There is always a vertex which is optimal (if there is an optimum at all)
- ▷ Idea of the Simplex Algorithm: Jump from vertex to vertex in the direction of the objective vector until an optimal vertex is reached
  - ➡ Details:

How to get to a neighbouring vertex?

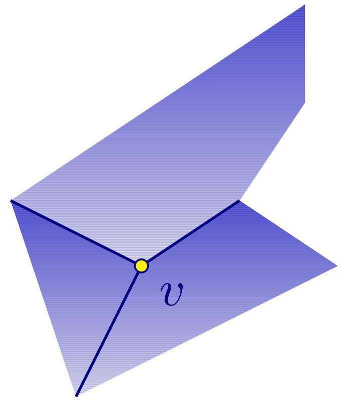
How to realise that the optimum is reached?

What if there is no optimum?

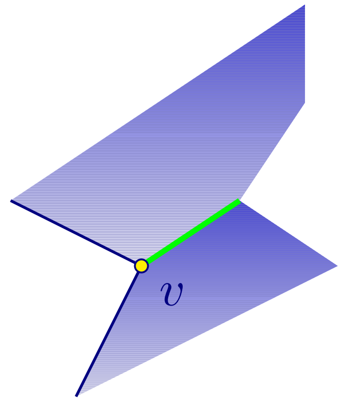
Or no feasible solution at all?!

* constraints have to be linearly independent!

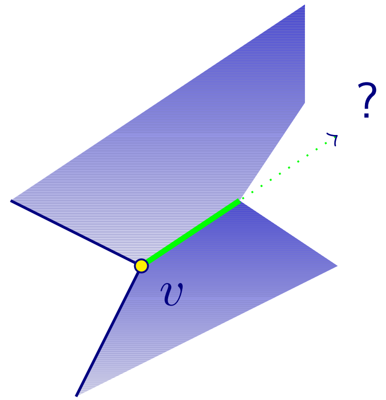
- Two vertices v, w are neighbours of each other if they are connected by an edge



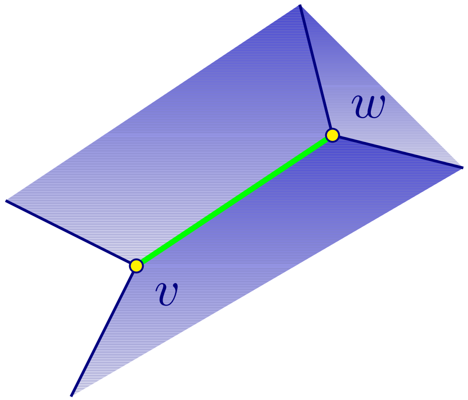
- ▷ Two vertices v, w are neighbours of each other if they are connected by an edge



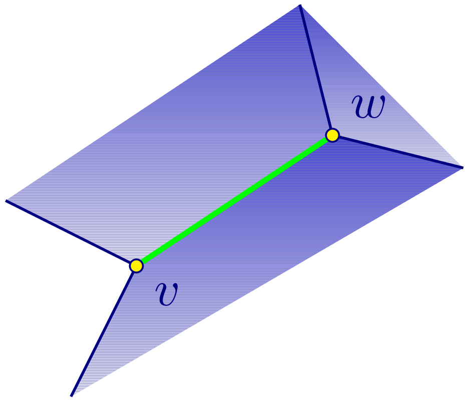
- Two vertices v, w are neighbours of each other if they are connected by an edge



- Two vertices v, w are neighbours of each other if they are connected by an edge

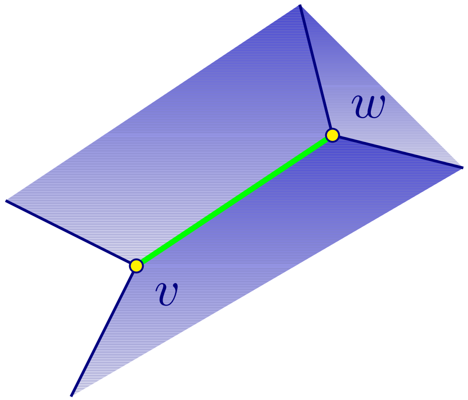


- ▷ Two vertices v, w are neighbours of each other if they are connected by an edge
  - ➡ To compute w from v only one binding constraint has to be exchanged!

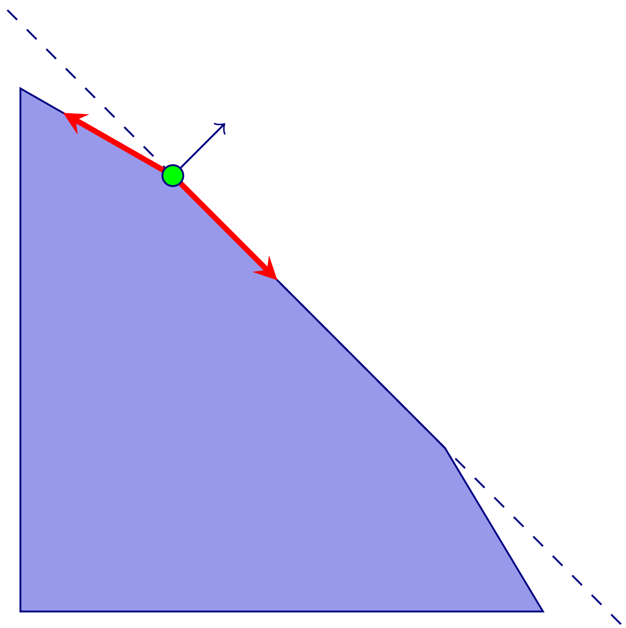


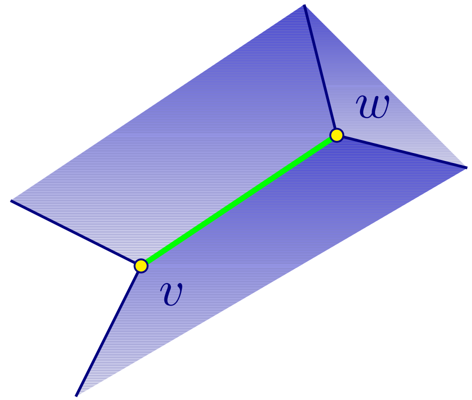
- Two vertices v, w are neighbours of each other if they are connected by an edge

  ➡ To compute w from v only one binding constraint has to be exchanged!

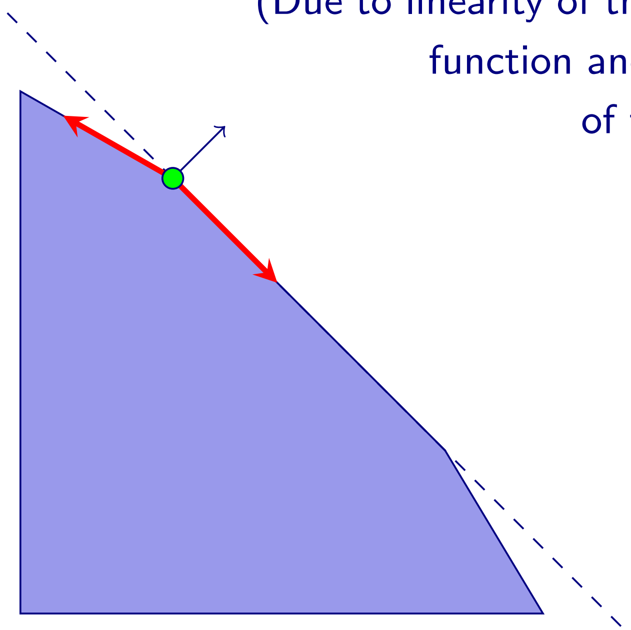


- If a vertex has no neighbours with a better objective function value then an optimum is reached!





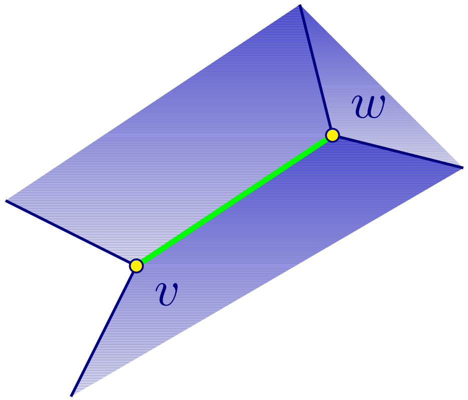
- Two vertices v, w are neighbours of each other if they are connected by an edge
  - ➡ To compute w from v only one binding constraint has to be exchanged!
- If a vertex has no neighbours with a better objective function value then an optimum is reached!



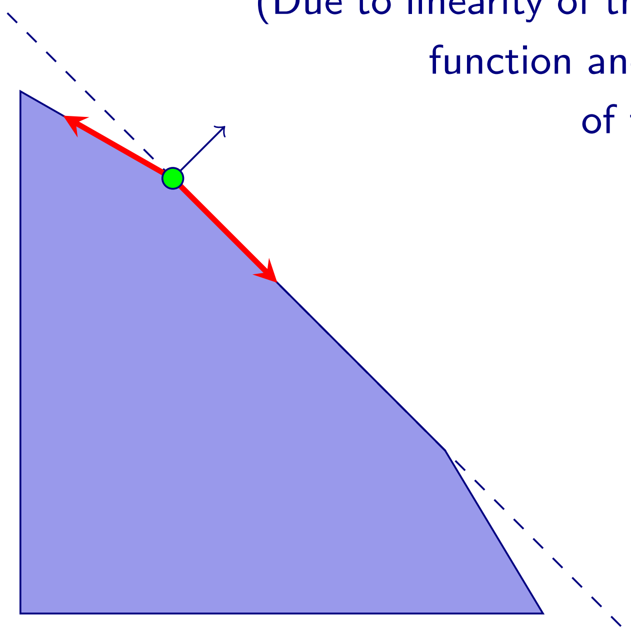
(Due to linearity of the objective function and convexity of the feasible region!)

▷ Two vertices v, w are neighbours of each other if they are connected by an edge

➡ To compute w from v only one binding constraint has to be exchanged!



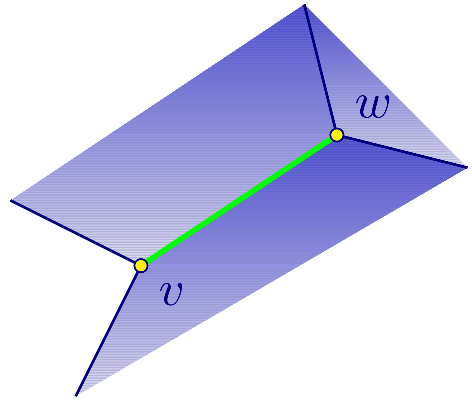
▷ If a vertex has no neighbours with a better objective function value then an optimum is reached!



(Due to linearity of the objective function and convexity of the feasible region!)

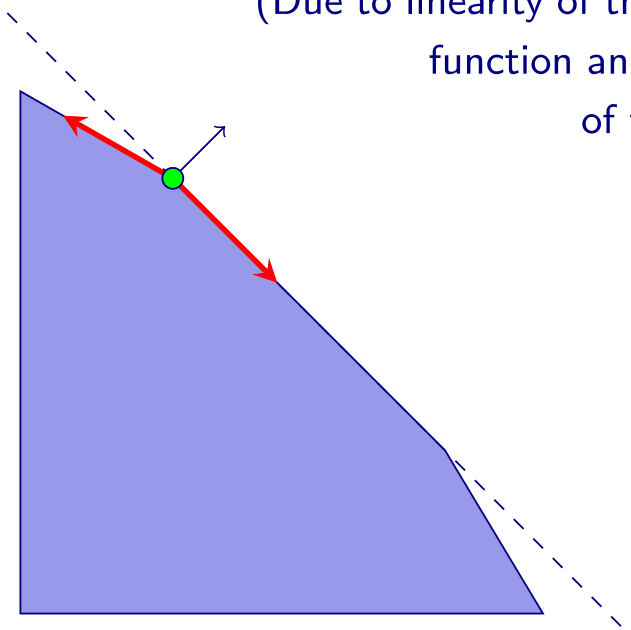
▷ If no opposite vertex can be computed (i.e. no “opposite” constraint found), then the problem is unbounded!

- Two vertices v, w are neighbours of each other if they are connected by an edge
  - ➡ To compute w from v only one binding constraint has to be exchanged!

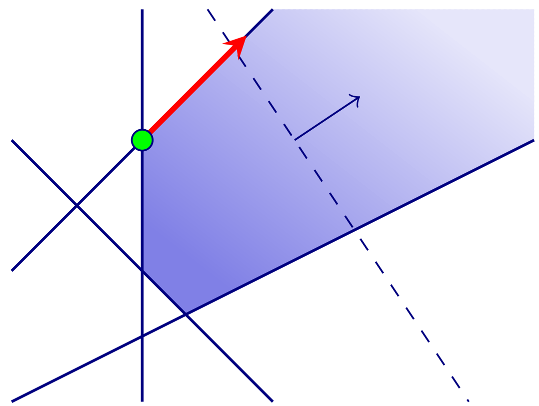


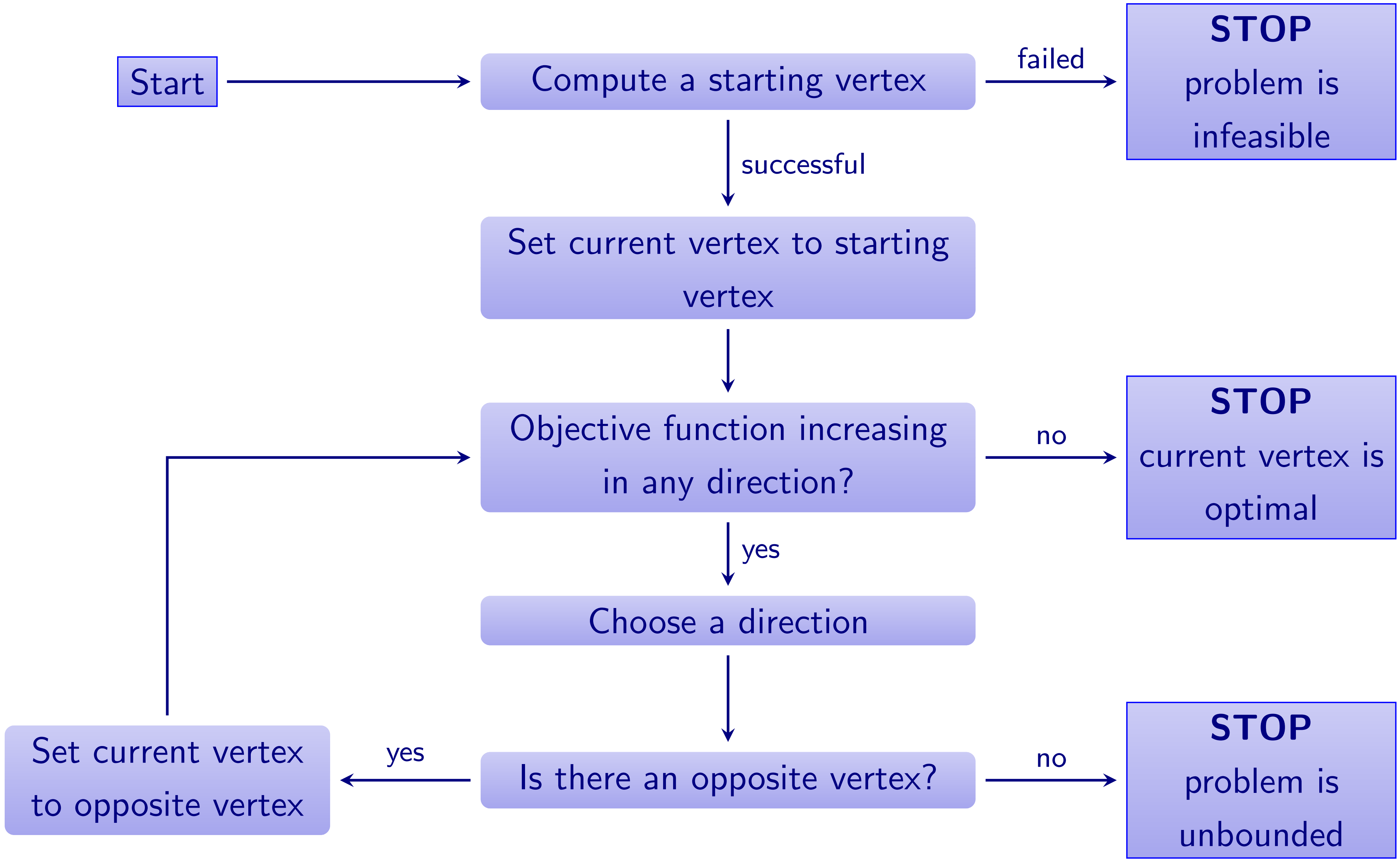
- If a vertex has no neighbours with a better objective function value then an optimum is reached!

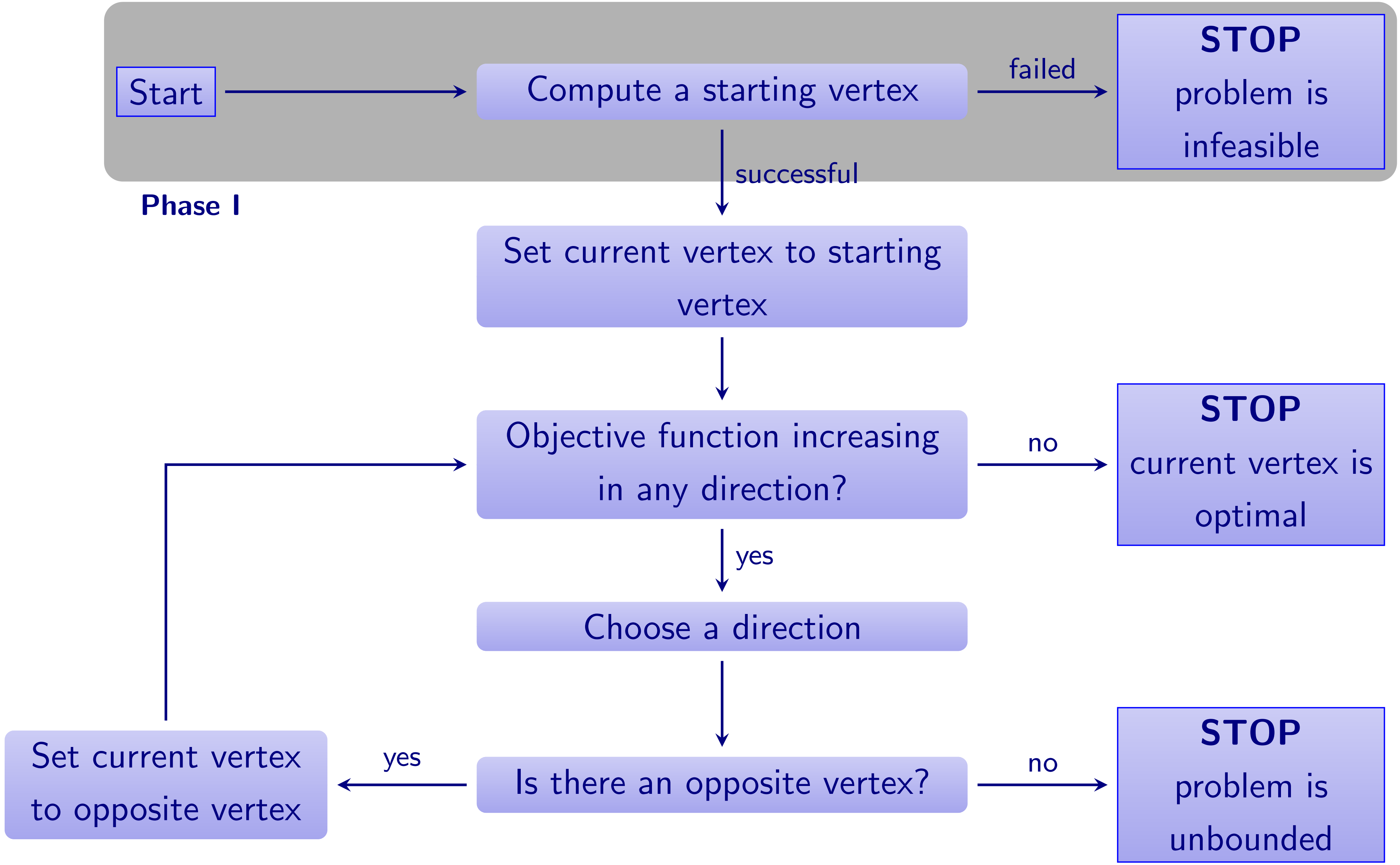
(Due to linearity of the objective function and convexity of the feasible region!)

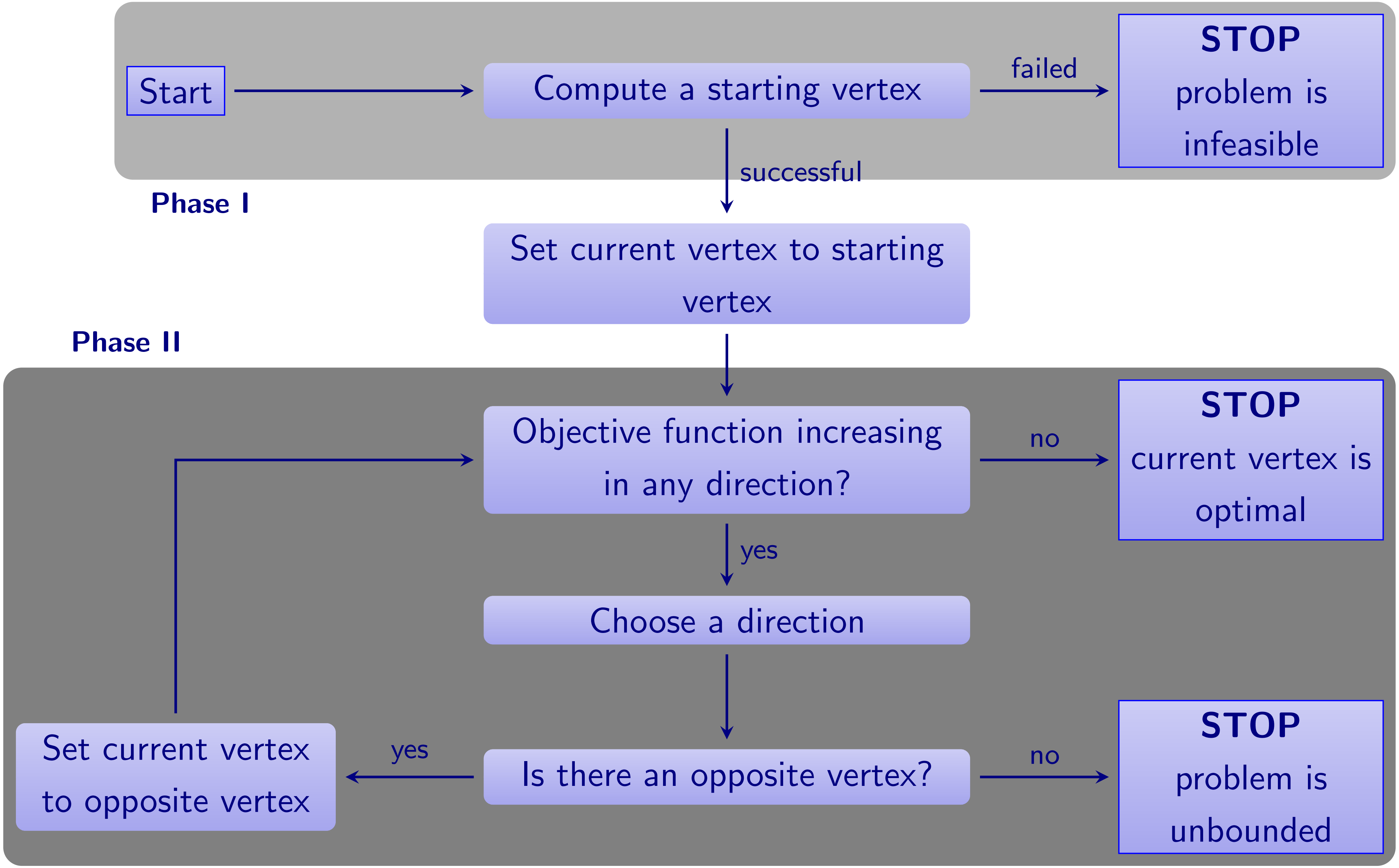


- If no opposite vertex can be computed (i.e. no “opposite” constraint found), then the problem is unbounded!




Start
Compute a starting vertex
failed
STOP
problem is infeasible
successful
Set current vertex to starting vertex
Objective function increasing in any direction?
no
STOP
current vertex is optimal
yes
Choose a direction
Is there an opposite vertex?
yes
Set current vertex to opposite vertex
no
STOP
problem is unbounded


Start
Compute a starting vertex
failed
STOP
problem is infeasible
successful
Phase I
Set current vertex to starting vertex
Objective function increasing in any direction?
no
STOP
current vertex is optimal
yes
Choose a direction
Set current vertex to opposite vertex
yes
Is there an opposite vertex?
no
STOP
problem is unbounded


Start
Compute a starting vertex
failed
STOP
problem is infeasible
successful
Phase I
Set current vertex to starting vertex
Phase II
Objective function increasing in any direction?
no
STOP
current vertex is optimal
yes
Choose a direction
Set current vertex to opposite vertex
yes
Is there an opposite vertex?
no
STOP
problem is unbounded

▷ Problem: Compute a starting vertex

- ▷ Problem: Compute a starting vertex

  ➡ Phase I: Formulate an auxiliary LP from the original with

▷ Problem: Compute a starting vertex

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex

▷ Problem: Compute a starting vertex

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

▷ Problem: Compute a starting vertex

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Compute a starting vertex**

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Choose a direction**

▷ Problem: **Compute a starting vertex**

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Choose a direction**

▷ Bad Pivot rules may lead to inefficient behaviour

▷ Problem: **Compute a starting vertex**

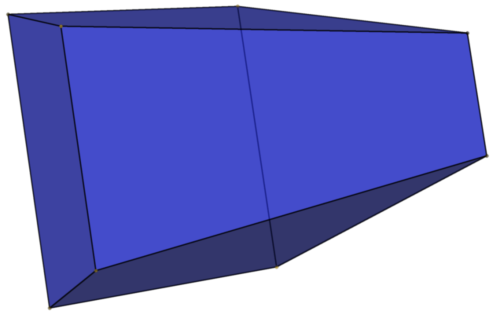
➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Choose a direction**

▷ Bad Pivot rules may lead to inefficient behaviour



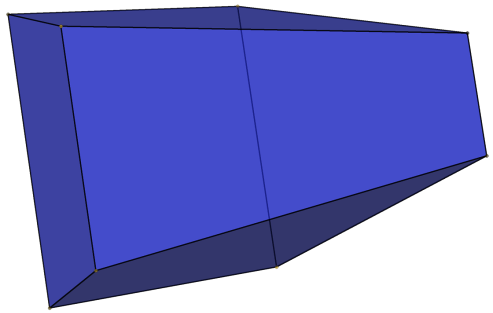
▷ Problem: **Compute a starting vertex**

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Choose a direction**



▷ Bad Pivot rules may lead to inefficient behaviour or even cycling (with degenerate problems)

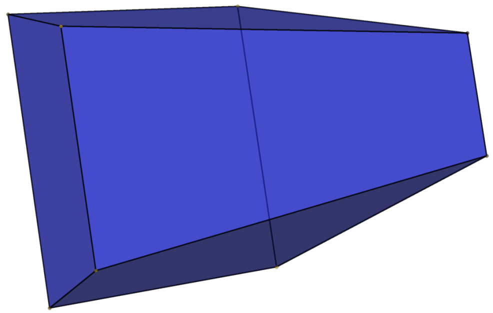
▷ Problem: Compute a starting vertex

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: Choose a direction



▷ Bad Pivot rules may lead to inefficient behaviour or even cycling (with degenerate problems)

▷ Numerical Problems: Computers can only compute with limited precision

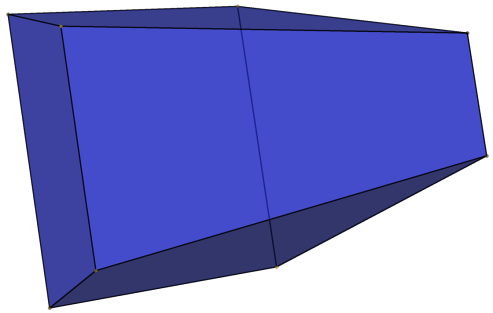
▷ Problem: Compute a starting vertex

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: Choose a direction



▷ Bad Pivot rules may lead to inefficient behaviour or even cycling (with degenerate problems)

▷ Numerical Problems: Computers can only compute with limited precision

➡ May lead to unprecise solution values, or even completely wrong solutions, or also cycling!

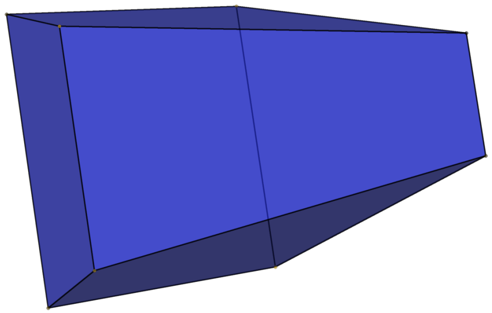
▷ Problem: **Compute a starting vertex**

➡ Phase I: Formulate an auxiliary LP from the original with

- easy to see starting vertex
- a solution of the auxiliary with optimal value 0 gives a starting vertex for the original

and solve it by using (Phase II) of the simplex algorithm itself!

▷ Problem: **Choose a direction**



▷ Bad Pivot rules may lead to inefficient behaviour or even cycling (with degenerate problems)

▷ Numerical Problems: Computers can only compute with limited precision

➡ May lead to unprecise solution values, or even completely wrong solutions, or also cycling! ➡ Use numerically stable computations!

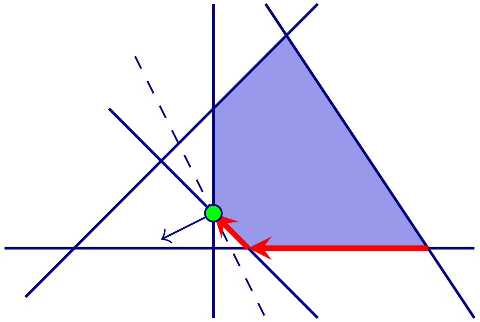
- A basic solution is primal feasible if is satisfies all constraints

- A basic solution is primal feasible if is satisfies all constraints
- A (primal feasible) basic solution is optimal if there is no neighbouring (primal feasible) basic solution with a better objective

- A basic solution is primal feasible if is satisfies all constraints
- A (primal feasible) basic solution is optimal if there is no neighbouring (primal feasible) basic solution with a better objective
- Dual feasible basic solutions have objective value at least as good as that of an optimal solution (but are not necessarily primal feasible)

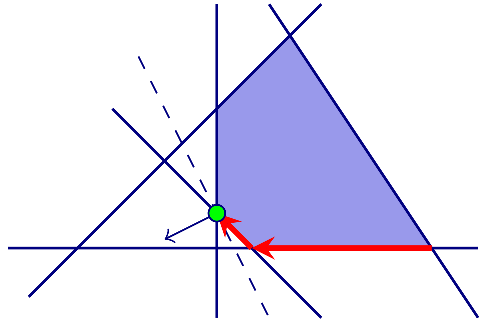
- A basic solution is primal feasible if is satisfies all constraints
- A (primal feasible) basic solution is optimal if there is no neighbouring (primal feasible) basic solution with a better objective
- Dual feasible basic solutions have objective value at least as good as that of an optimal solution (but are not necessarily primal feasible)

(Primal) Simplex Algorithm: Step from primal feasible solution to neighbouring primal feasible solution in direction of the objective until an optimal solution is reached

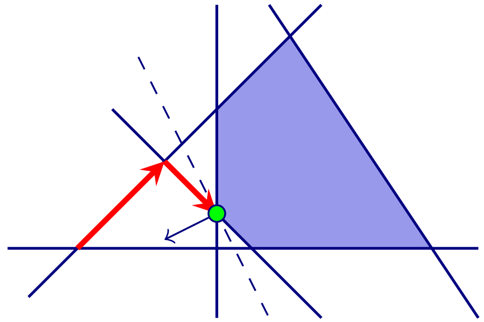


- A basic solution is primal feasible if is satisfies all constraints
- A (primal feasible) basic solution is optimal if there is no neighbouring (primal feasible) basic solution with a better objective
- Dual feasible basic solutions have objective value at least as good as that of an optimal solution (but are not necessarily primal feasible)

(Primal) Simplex Algorithm: Step from primal feasible solution to neighbouring primal feasible solution in direction of the objective until an optimal solution is reached

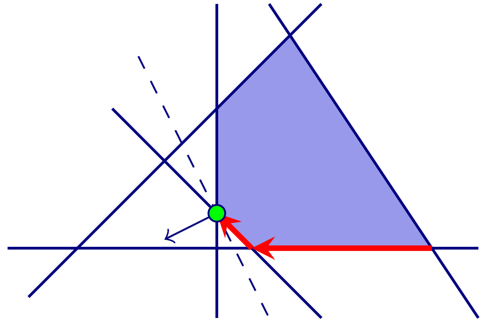


Dual Simplex Algorithm: Step from dual feasible solution to neighbouring dual feasible solution until a primal feasible solution is reached (which is then also optimal!)

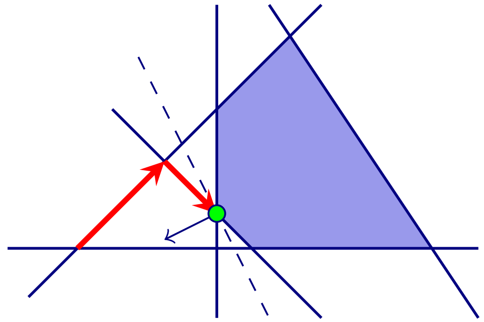


- A basic solution is primal feasible if is satisfies all constraints
- A (primal feasible) basic solution is optimal if there is no neighbouring (primal feasible) basic solution with a better objective
- Dual feasible basic solutions have objective value at least as good as that of an optimal solution (but are not necessarily primal feasible)

(Primal) Simplex Algorithm: Step from primal feasible solution to neighbouring primal feasible solution in direction of the objective until an optimal solution is reached



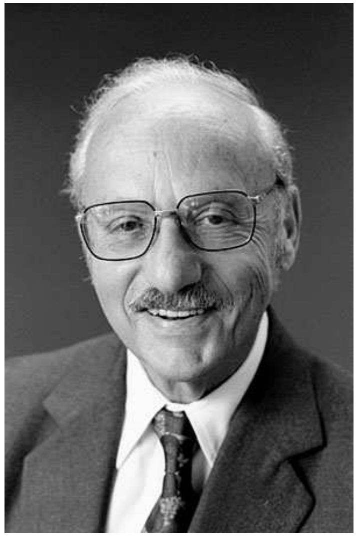
Dual Simplex Algorithm: Step from dual feasible solution to neighbouring dual feasible solution until a primal feasible solution is reached (which is then also optimal!)



- Dual simplex seems to be much more efficient (on average on real-world problems)!

- ▷ Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947

- Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947



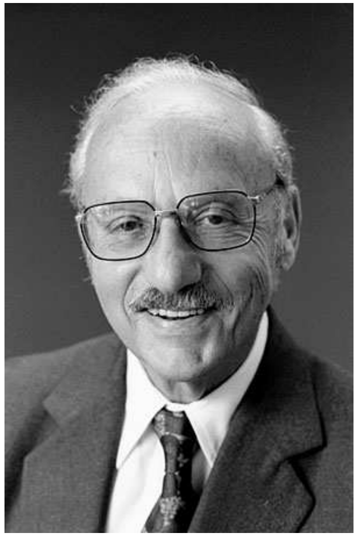
George Bernard Dantzig
(1914–2005)

▷ Simplex Algorithm

➡ developed by George B. Dantzig in 1947

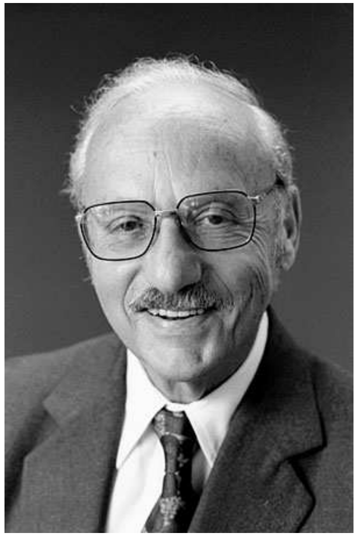
➡ Variants:

- Dual Simplex Algorithm



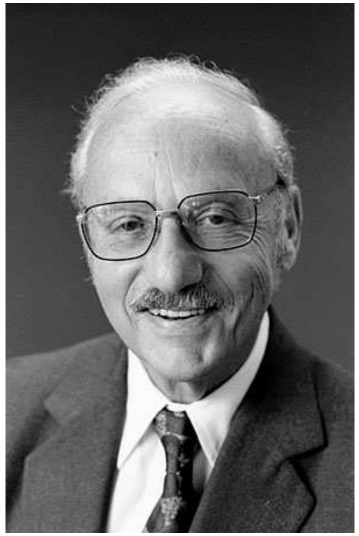
George Bernard Dantzig
(1914–2005)

- Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex



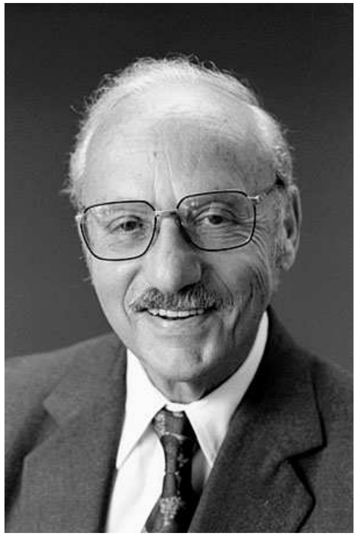
George Bernard Dantzig
(1914–2005)

- ▷ Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex
- ▷ Ellipsoid Method

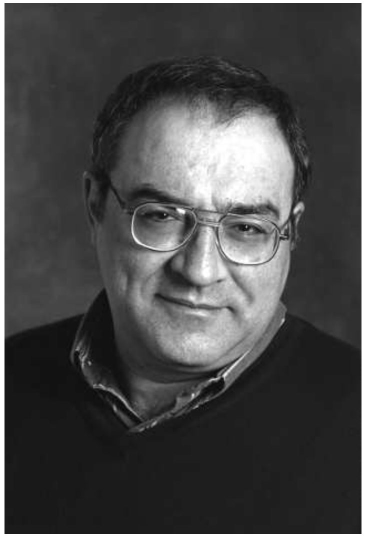


George Bernard Dantzig
(1914–2005)

- Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex
- Ellipsoid Method
  - ➡ developed by L.G. Khachiyan in 1979

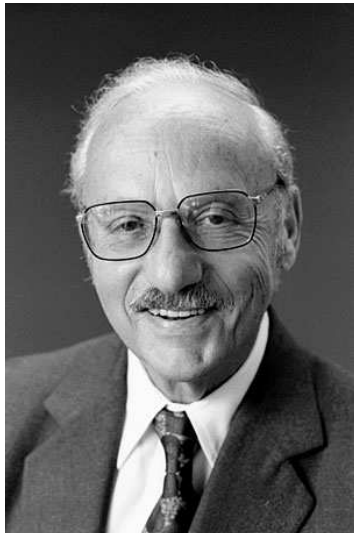


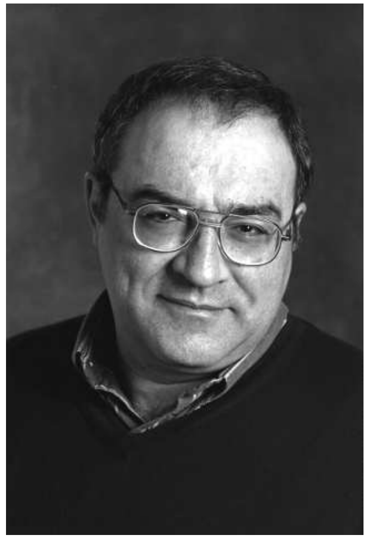
George Bernard Dantzig
(1914–2005)



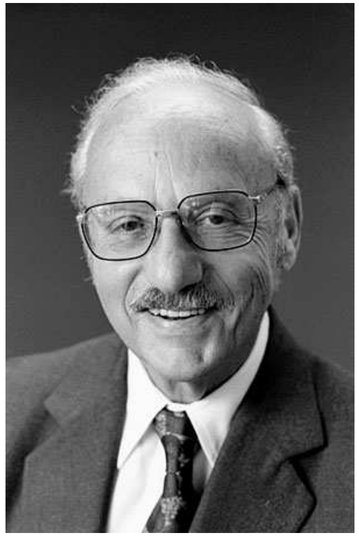
Leonid Genrikhovich Khachiyan
(1952–2005)

- Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex
- Ellipsoid Method
  - ➡ developed by L.G. Khachiyan in 1979
  - ➡ theoretically fast (polynomial)

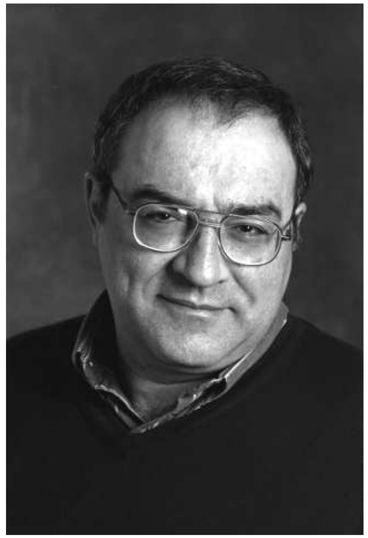

George Bernard Dantzig
(1914–2005)


Leonid Genrikhovich Khachiyan
(1952–2005)

- ▷ Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex
- ▷ Ellipsoid Method
  - ➡ developed by L.G. Khachiyan in 1979
  - ➡ theoretically fast (polynomial), but practically useless



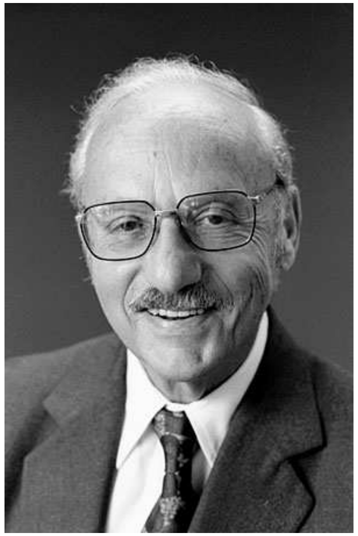
George Bernard Dantzig
(1914–2005)



Leonid Genrikhovich Khachiyan
(1952–2005)

▷ Simplex Algorithm

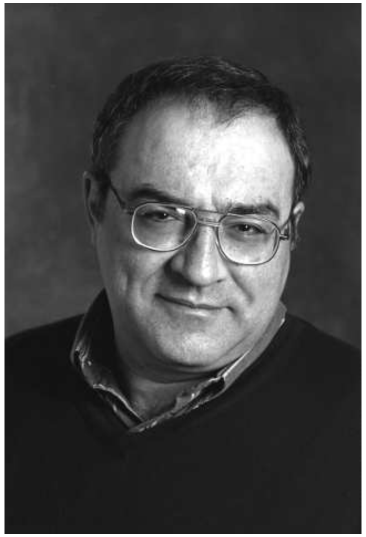
- ➡ developed by George B. Dantzig in 1947
- ➡ Variants:
  - Dual Simplex Algorithm
  - Network Simplex



George Bernard Dantzig
(1914–2005)

▷ Ellipsoid Method

- ➡ developed by L.G. Khachiyan in 1979
- ➡ theoretically fast (polynomial), but practically useless



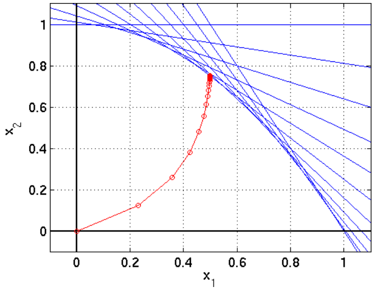
Leonid Genrikhovich Khachiyan
(1952–2005)

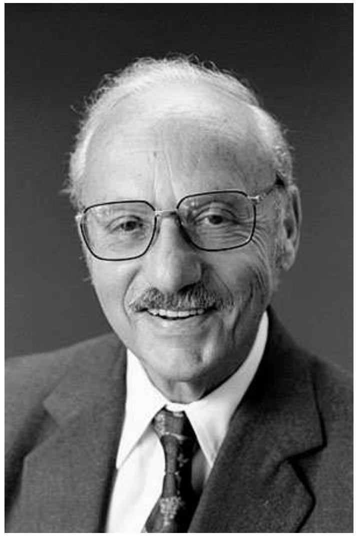
▷ Interior Point Methods

- ➡ Barrier Method (Karmarkar, 1984)

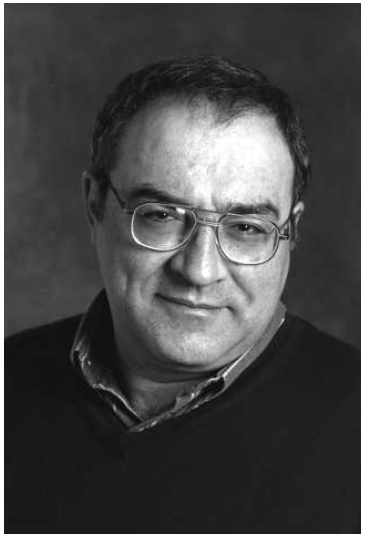
LP algorithm zoo

- ▷ Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex
- ▷ Ellipsoid Method
  - ➡ developed by L.G. Khachiyan in 1979
  - ➡ theoretically fast (polynomial), but practically useless
- ▷ Interior Point Methods
  - ➡ Barrier Method (Karmarkar, 1984)





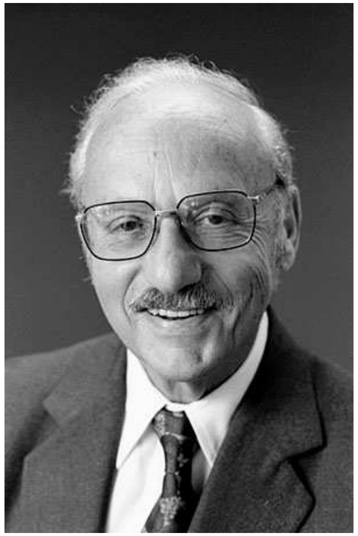
George Bernard Dantzig
(1914–2005)



Leonid Genrikhovich Khachiyan
(1952–2005)

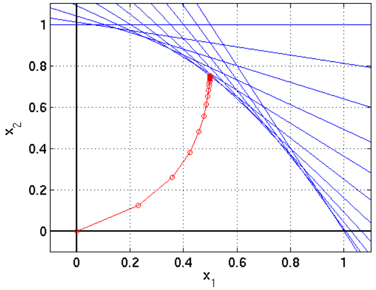
LP algorithm zoo

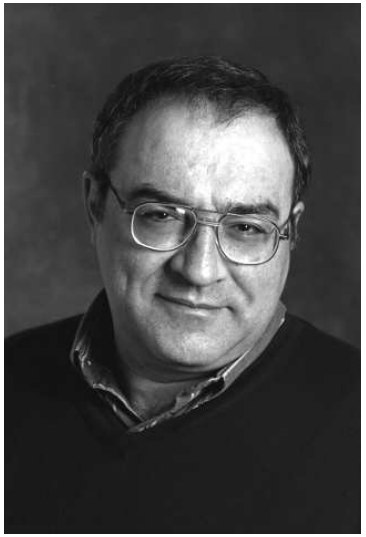
- ▷ Simplex Algorithm
  - ➡ developed by George B. Dantzig in 1947
  - ➡ Variants:
    - Dual Simplex Algorithm
    - Network Simplex



George Bernard Dantzig
(1914–2005)

- ▷ Ellipsoid Method
  - ➡ developed by L.G. Khachiyan in 1979
  - ➡ theoretically fast (polynomial), but practically useless
- ▷ Interior Point Methods
  - ➡ Barrier Method (Karmarkar, 1984)
  - ➡ theoretically and practically fast
  - ➡ used for large-scale LPs





Leonid Genrikhovich Khachiyan
(1952–2005)

▷ LP solving in 1947*:

Perhaps the first instance of a nontrivial LP solved with the simplex algorithm was Laderman's solution (see Dantzig 1963) of Stigler's (1945) diet problem. This LP had nine constraints and 77 variables. Reportedly, nine coworkers working on electronic calculators for an estimated total of 120 man-days were needed to carry out the computations.

*from: Bixby, *Solving real-world linear programs: a decade and more of progress*, OR 50 (2002), 3–15

▷ LP solving in 1947*:

> Perhaps the first instance of a nontrivial LP solved with the simplex algorithm was Laderman's solution (see Dantzig 1963) of Stigler's (1945) diet problem. This LP had nine constraints and 77 variables. Reportedly, nine coworkers working on electronic calculators for an estimated total of 120 man-days were needed to carry out the computations.

▷ Improvements due to computer power (1987–2000)*:

Sun 3/50 vs Pentium 4, 1.7 GHz ➡ speedup factor 800

*from: Bixby, *Solving real-world linear programs: a decade and more of progress*, OR 50 (2002), 3–15

▷ LP solving in 1947*:

Perhaps the first instance of a nontrivial LP solved with the simplex algorithm was Laderman's solution (see Dantzig 1963) of Stigler's (1945) diet problem. This LP had nine constraints and 77 variables. Reportedly, nine coworkers working on electronic calculators for an estimated total of 120 man-days were needed to carry out the computations.

▷ Improvements due to computer power (1987–2000)*:

Sun 3/50 vs Pentium 4, 1.7 GHz ➡ speedup factor 800

▷ Improvements due to algorithms (1987–2000)*:

primal simplex 1988 vs primal/dual/barrier 2000 ➡ speedup factor 2360

➡ Total speedup: ≈ 1900000 times (1987–2000)

*from: Bixby, *Solving real-world linear programs: a decade and more of progress*, OR 50 (2002), 3–15

- LP solving in 1947*:

  Perhaps the first instance of a nontrivial LP solved with the simplex algorithm was Laderman's solution (see Dantzig 1963) of Stigler's (1945) diet problem. This LP had nine constraints and 77 variables. Reportedly, nine coworkers working on electronic calculators for an estimated total of 120 man-days were needed to carry out the computations.

- Improvements due to computer power (1987–2000)*:

  Sun 3/50 vs Pentium 4, 1.7 GHz ➡ speedup factor 800

- Improvements due to algorithms (1987–2000)*:

  primal simplex 1988 vs primal/dual/barrier 2000 ➡ speedup factor 2360

➡ Total speedup: $\approx$ 1900000 times (1987–2000)

- Conclusion (as of 2002)*:

  A model that might have taken a year to solve 10 years ago can now solve in less than 30 seconds.

*from: Bixby, *Solving real-world linear programs: a decade and more of progress*, OR 50 (2002), 3–15

- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
- ▷ (Mixed) Integer Programming
- ▷ Mathematical Background: Cuts, Branch & Bound
- ▷ Combinatorial Optimization
- ▷ Mathematical Background: Graphs, Algorithms
- ▷ Complexity Theory
- ▷ Nonlinear Optimization
- ▷ Scheduling
- ▷ Lot Sizing
- ▷ Multicriteria Optimization

- ▷ Exam

- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
  - ▷ Sensitivity Analysis
- ▷ (Mixed) Integer Programming
- ▷ Mathematical Background: Cuts, Branch & Bound
- ▷ Combinatorial Optimization
- ▷ Mathematical Background: Graphs, Algorithms
- ▷ Complexity Theory
- ▷ Nonlinear Optimization
- ▷ Scheduling
- ▷ Lot Sizing
- ▷ Multicriteria Optimization

- ▷ Exam