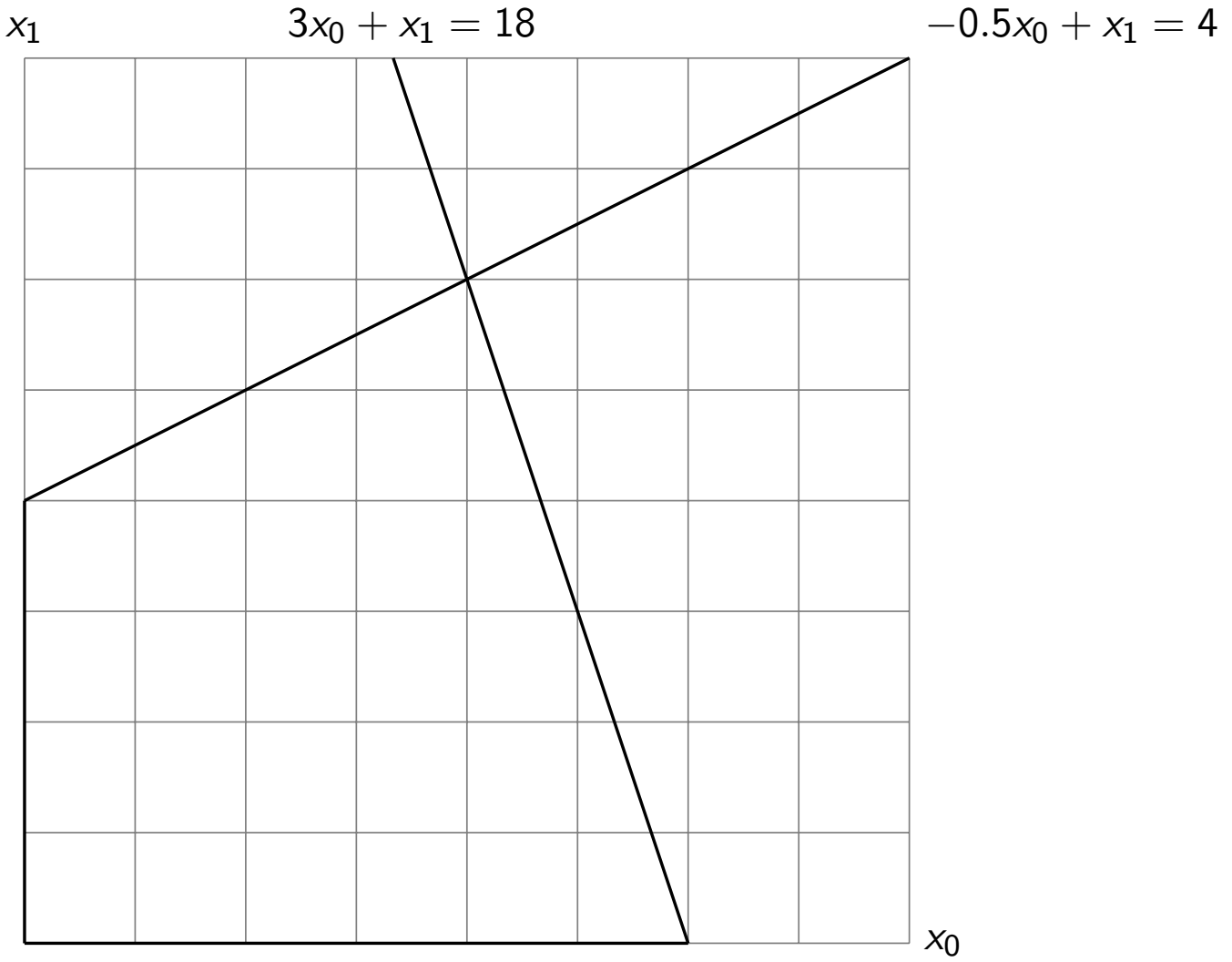
Contents Lecture 10

- Linear programming
- The simplex algorithm
- Integer linear programming
- Branch-and-bound
- Examples

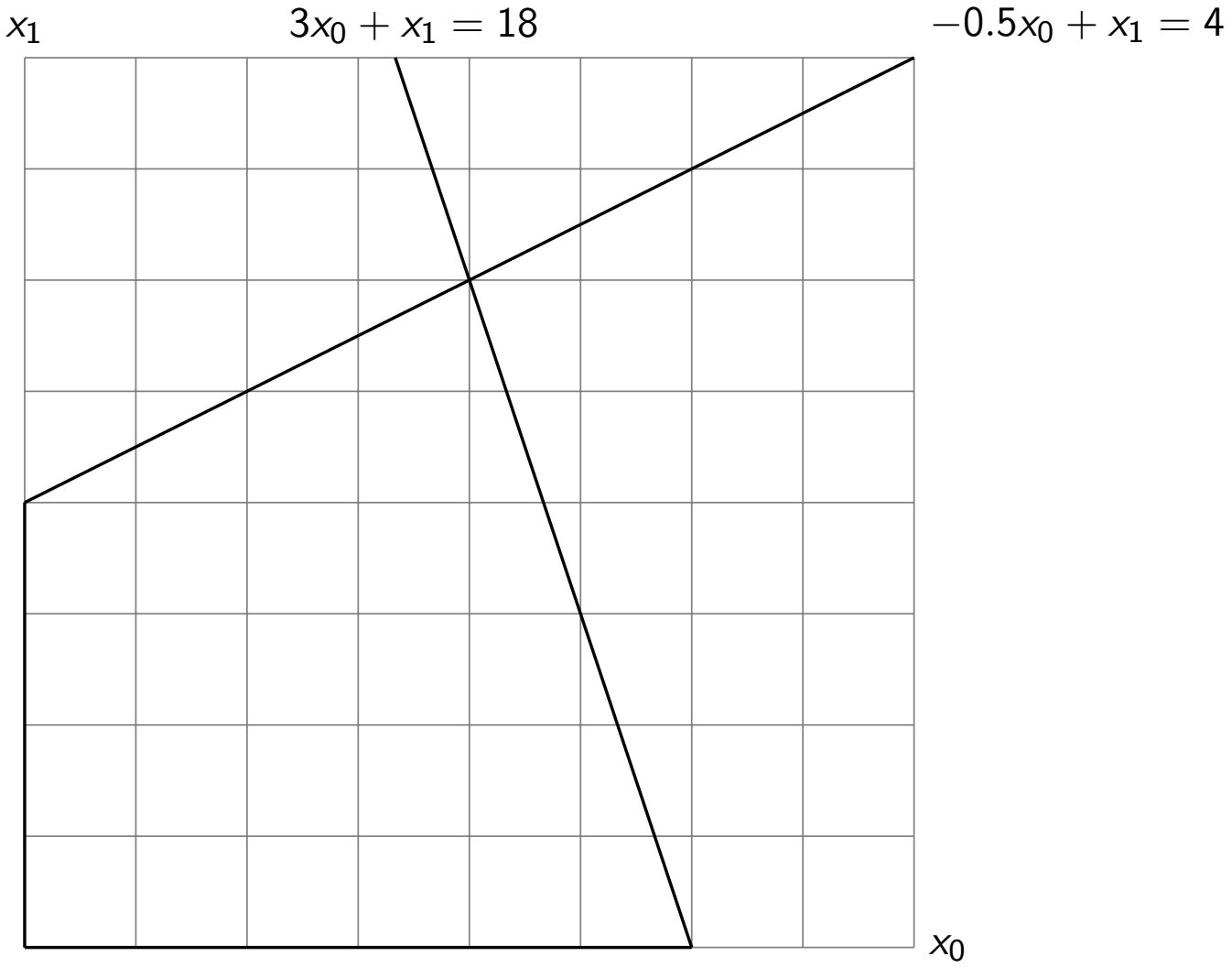
Linear program: maximize a linear function in a region

- a region defined by lines including $x_i \geq 0$, i.e. linear constraints, and
- a linear objective function such as $\max z = x_0 + 2x_1$



Solving a linear program

- Find $x_i \in \mathbb{R}$ which maximize z
- Here x_i are called decision variables



Our example

- Objective function and linear constraints using $\leq$

$$\begin{aligned} \max \quad z \quad &= x_0 + 2x_1 \\ \\ -0.5x_0 \quad + \quad x_1 \quad &\leq \quad 4 \\ 3x_0 \quad + \quad x_1 \quad &\leq \quad 18. \end{aligned}$$

- Also: implicitly $x_i \geq 0$
- We can also use $\geq$ and $=$ by first rewriting them to use $\leq$
- (with two $\leq$ instead of one $=$)

Linear programs

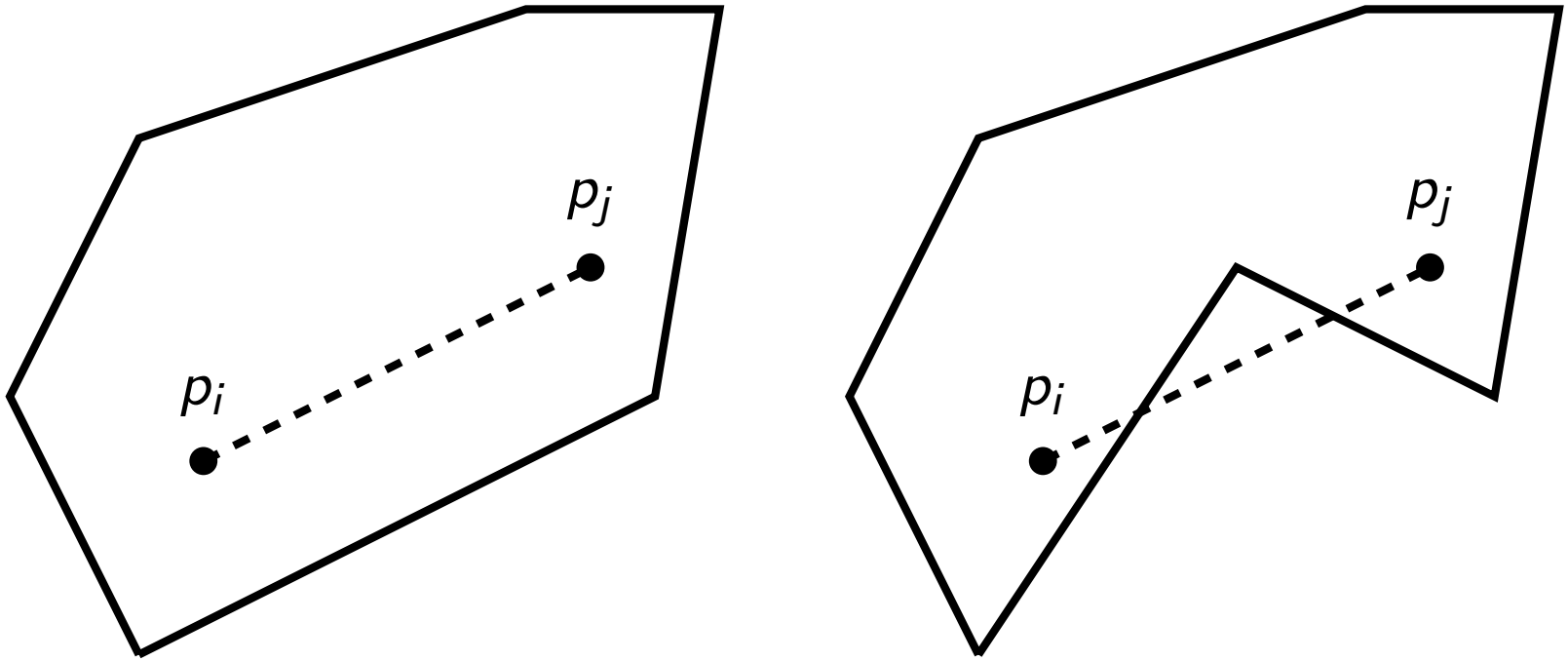
- $$\begin{aligned} \max \quad & z = c_0 x_0 + c_1 x_1 + \ldots + c_{n-1} x_{n-1} \\ & \\ & a_{0,0} x_0 + a_{0,1} x_1 + \ldots + a_{0,n-1} \leq b_0 \\ & a_{1,0} x_0 + a_{1,1} x_1 + \ldots + a_{1,n-1} \leq b_1 \\ & \ldots \\ & a_{m-1,0} x_0 + a_{m-1,1} x_1 + \ldots + a_{m-1,n-1} \leq b_{m-1} \\ & \\ & x_0, x_1, \ldots, x_{n-1} \geq 0 \end{aligned}$$

- or simpler as

$$\begin{aligned} \max \quad & z = \mathrm{c}\mathrm{x} \\ & \mathrm{A}\mathrm{x} \leq \mathrm{b} \\ & \mathrm{x} \geq 0. \end{aligned}$$

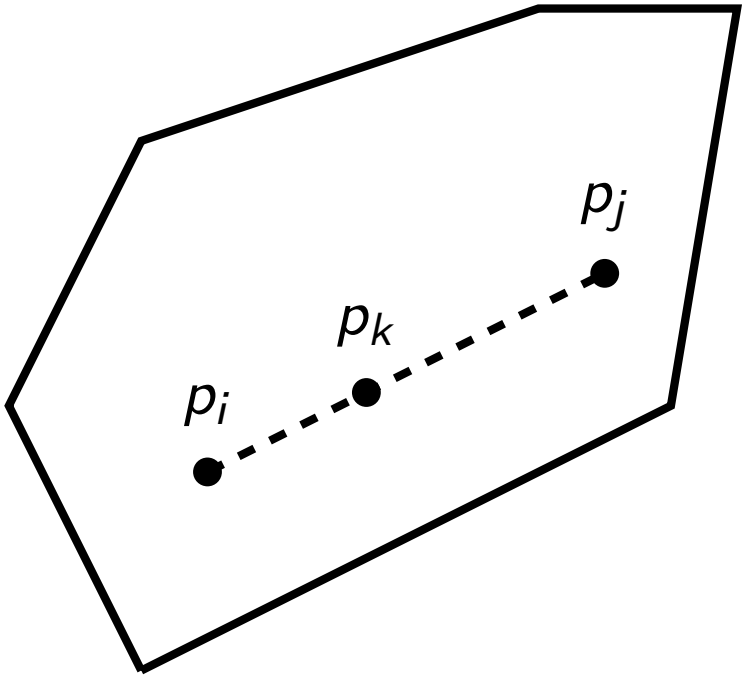
Solutions

- Each constraint defines a halfplane in n dimensions.
- The intersection of these halfplanes defines the **feasible region**, P, with **feasible solutions** $x \in P$.
- The feasible region is convex, and a point where halfplanes intersect is called a **vertex**.
- A non-convex region cannot be an intersection of halfplanes.



Consequence of P being a convex region

- Assume p_k is on a line segment through p_i and p_j
- We can write a point as $p_k = \lambda \cdot p_i + (1 - \lambda) \cdot p_j$, with $0 \leq \lambda \leq 1$
- So that p_k is in the region
- Here $\lambda = 0.6$



Solutions

- A linear program is either:
  - **infeasible** when P is empty,
  - **unbounded** when no finite solution exists, or
  - **feasible**, in which case we search for an optimal solution $x^* \in P$ which maximizes z.
- There may exist more than one optimal solution.

Local and global optimal solutions

- We denote by $z(\mathrm{x})$ the value of the objective function z at point x.
- A solution x is a **local optimum** for $z(\mathrm{x})$ if there is an $\epsilon > 0$ such that $z(\mathrm{x}) \geq z(\mathrm{y})$ for all $\mathrm{y} \in \mathrm{P}$ with $||x - y|| \leq \epsilon$.

Theorem

A local optimum of a linear program is also a global optimum.

Theorem

For a bounded feasible linear program with feasible region P, at least one vertex is an optimal solution.

- So we only need to check z in the vertices and not the inner part of the region.

A local optimum of a linear program is also a global optimum

Proof.

- $z(u)$ is a linear objective function
- Assume $z(u) > z(v)$ for all v at most ϵ from u
- Let w be any point in P, possibly far away from u and v
- $v = \lambda u + (1 - \lambda)w$ with $0 \leq \lambda \leq 1$
- $z(u) \geq z(v) = z(\lambda u + (1 - \lambda)w) \geq \lambda z(u) + (1 - \lambda)z(w)$
- So $z(u) - \lambda z(u) \geq (1 - \lambda)z(w)$
- And $z(u)(1 - \lambda) \geq (1 - \lambda)z(w)$
- $0 \leq \lambda \leq 1$, so $z(u) \geq z(w)$

At least one vertex is an optimal solution

Proof.

- Let the feasible region P have k vertices: $\mathrm{x}^0, \mathrm{x}^1, ..., \mathrm{x}^{\mathrm{k}-1}$
- Let z^* be the maximum value in any vertex: $z^* = \max\{\mathrm{cx}^{\mathrm{i}}, 0 \leq i < k\}$
- Every point w in P can be written as a linear combination of the vertices: $w = \sum_{i=0}^{k-1} \lambda_i \mathrm{x}^{\mathrm{j}}$ with $\sum_{i=0}^{k-1} \lambda_i = 1$
- Let w be any point in P
- $z(\mathrm{w}) = \mathrm{cw} = \mathrm{c} \sum \lambda_i \mathrm{x}^{\mathrm{i}} = \sum \lambda_i (\mathrm{cx}^{\mathrm{i}}) \leq \sum \lambda_i z^* = z^* \sum \lambda_i = z^*$.

Slack form

$$
\begin{aligned}
\max \quad & cx \\
x_{n+0} &= b_0 - \textstyle\sum_{j=0}^{n-1} a_{0,j} x_j \\
x_{n+1} &= b_1 - \textstyle\sum_{j=0}^{n-1} a_{1,j} x_j \qquad (1) \\
&\dots \\
x_{n+m-1} &= b_{m-1} - \textstyle\sum_{j=0}^{n-1} a_{m-1,j} x_j \\
x_i &\geq 0 \qquad 0 \leq i \leq n+m-1
\end{aligned}
$$

- The variables on the left hand side are called **basic variables** and occur only once, i.e. neither in any sum on the right hand side, nor in the objective function.
- The other variables are called **nonbasic variables**.

Slack form of our example

- We start with

$$\begin{aligned} \max \quad z \quad &= x_0 + 2x_1 \\ \\ -0.5x_0 \quad + \quad x_1 \quad &\leq \quad 4 \\ 3x_0 \quad + \quad x_1 \quad &\leq \quad 18 \end{aligned}$$

- and then introduce two new variables, one for each constraint, and write it on slack form:

$$\begin{aligned} \max \quad z \quad &= x_0 + 2x_1 + y \\ \\ x_2 \quad = \quad 4 \quad - \quad (\quad -0.5x_0 \quad + \quad x_1 \quad) \\ x_3 \quad = \quad 18 \quad - \quad (\quad 3x_0 \quad + \quad x_1 \quad). \end{aligned}$$

- All $x_i \geq 0$ and y is initially zero
- We rewrite the problem until all coefficients in the objective function become negative, and set all nonbasic variables to zero

Entering and leaving basic variables

- Select a nonbasic variable with positive c_i coefficient
- We take nonbasic variable x_0 as the so called **entering basic variable**

$$\max \quad z \quad = x_0 + 2x_1 + y$$

$$\begin{aligned} x_2 &= 4 - (-0.5x_0 + x_1) \\ x_3 &= 18 - (3x_0 + x_1). \end{aligned}$$

- Since c_0 is positive, we want to increase x_0 as much as possible
- The basic variables can limit how much x_0 may be increased (if there is no restriction, then the linear program is unbounded)
- x_3 restricts increasing x_0 to at most 6.
- Therefore we select x_3 as the so called **leaving basic variable**.

Rewritten linear program

- We rewrite the linear program by letting the entering and leaving basic variables switch roles.
- This is a tedious but simple algebraic manipulation
- Do this by hand at least once

$$\max \quad z \quad = -0.333x_3 + 1.667x_1 + 6$$

$$\begin{aligned} x_2 &= 7 - (\ 0.167x_3 + 1.167x_1\ ) \\ x_0 &= 6 - (\ 0.333x_3 + 0.333x_1\ ) \end{aligned}$$

- Next we must select x_1 as entering basic variable
- x_2 is restricted by $7 - 1.167x_1 \geq 0$
- x_0 is restricted by $6 - 0.333x_1 \geq 0$
- x_2 is most restricted and becomes the leaving basic variable

Solution

- All c_i are negative so z cannot be increased with positive values of the nonbasic variables.
- By setting the nonbasic variables to zero the maximum becomes 16 in $x = (4, 6)$ which indeed is a vertex.

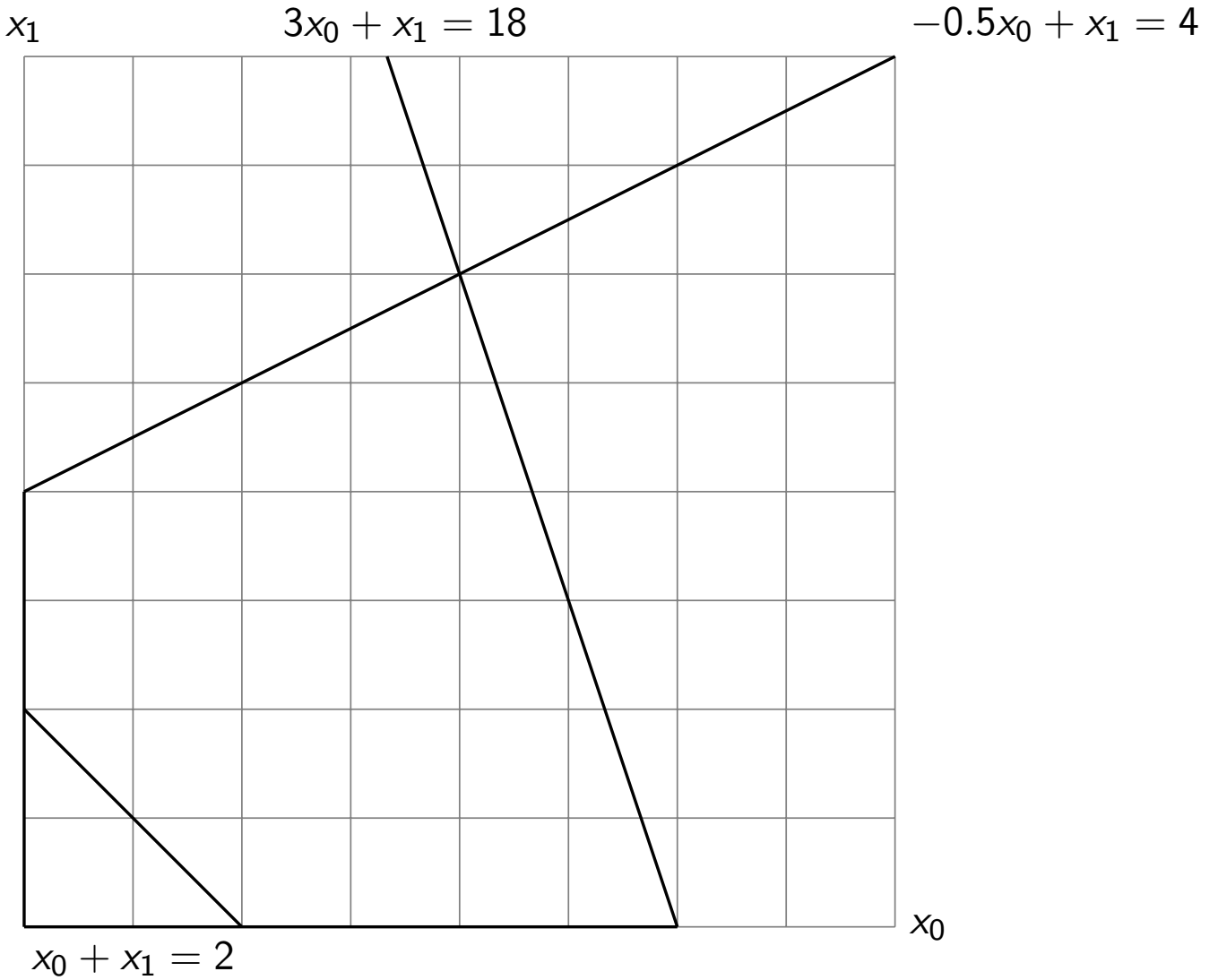
$$\max \quad z \quad = -0.6x_3 - 1.4x_2 + 16$$

$$\begin{aligned} x_1 &= 6 - (\ 0.1x_3 + 0.9x_2\ ) \\ x_0 &= 4 - (\ 0.3x_3 - 0.3x_2\ ) \end{aligned}$$

- Summary: we start in a vertex and then go to a neighboring vertex until all coefficients are negative, which gives the optimal solution.
- It was an open problem but George Dantzig was late for a lecture at Berkeley and mistook it for a home assignment (he got a PhD for it).

A problem

- We cannot start simplex in $(0, 0)$ if it is not in the feasible region



Finding a start vertex when there is a negative b_i

- Let our original linear program be P_0
- If some b_i is negative the point 0 is not in the feasible region
- We create a new problem P_1 to find a start vertex for P_0
- Add a new nonbasic variable x_{n+m}
- Start with P_0 and subtract x_{n+m} from each constraint:

$$a_{i,0}x_0 + a_{i,1}x_1 + \ldots + a_{i,n-1} - x_{n+m} \quad \leq \quad b_i$$

- Use the objective function $z_1 = -x_{n+m}$
- We do a pivot on P_1 with x_{m+n} as entering basic variable and x_k with most negative b_k as leaving basic variable
- This gives us b ≥ 0, so P_1 can be solved using the simplex algorithm,
- If P_1 has optimal value 0 then $x_{n+m} = 0$ and by removing x_{m+n} from this solution, we have a start vertex for P_0

Integer programming

- Integer programming is similar to linear programming with the extra constraint that $x_i \in \mathbb{N}$.
- Some problems including this have no efficient algorithms
- One bad "method" to solve problems is to enumerate all solutions
- This does not sound good though
- We will use the algorithm design paradigm branch-and-bound to solve integer programs (not all due to integer programming is NP-complete)

Branch

- A **relaxation** makes a problem simpler (by solving another problem)
- Perfekt tentasvar: "man softar problemet".
- For integer programming we solve the corresponding linear program, i.e. relaxing the integer requirement on the solution.
- Suppose we have an integer program and give it to the simplex algorithm and $x_k \notin \mathbb{Z}$
- Assume the simplex algorithm assigns $x_k = u$
- We can then branch by creating two new linear programs:
  - one with the additional constraint constraint $x_k \leq \lfloor u \rfloor$, and
  - another with the additional constraint $x_k \geq \lceil u \rceil$.
- Each new problem is solved directly with the simplex algorithm
- If it has an integer solution we can limit the search tree (bound)
- If it has a non-integer solution and it is better than the best integer solution we put it in a queue

Bound

- If the simplex algorithm found an integer solution we do the following
- We check if this is the best integer solution found so far, and remember it in that case
- We remove from the queue all unexplored linear programs whose optimal value is less than the value of the integer solution we just found

Graph coloring

- Consider an undirected graph $G(V, E)$ with m edges and n vertices.
- At most n colors are needed and we create n binary decision variables w_i for $0 \leq i < n$, with the meaning $w_i = 0$ if color i is unused and $w_i = 1$ if at least one vertex is assigned color i.
- For each vertex v we then create n decision variables x_{vi}, also binary, with the meaning that $x_{vi} = 1$ if vertex v is assigned color i and otherwise zero. The object function is to minimize the number of used colors, i.e., the sum of all w_i. There are two types of constraints:
  - each vertex must be assigned a color, so the sum of all x_{vi} for a particular vertex v must be one, and
  - two neighbors cannot use the same color, so if u and v are neighbors $x_{ui} + x_{vi} \leq w_i$, for all i.

First model

The model becomes:

$$\min \quad z \quad = w_0 + w_1 + \ldots + w_{n-1}$$

$$\sum_{i=0}^{n-1} x_{vi} = 1 \qquad \forall v \in V$$

$$x_{ui} + x_{vi} \leq w_i \qquad \forall (u, v) \in E$$

$$x_{vi}, w_i \in \{0, 1\} \qquad \forall v \in V, 0 \leq i < n$$

Second model

Since we expect the constraints to be on the form $Ax \leq b$, we write it as follows instead:

$$\min \quad z \quad = w_0 + w_1 + ... + w_{n-1}$$

$$\begin{aligned}
\sum_{i=0}^{n-1} x_{vi} &\leq 1 \quad \forall v \in V \\
\sum_{i=0}^{n-1} -x_{vi} &\leq -1 \quad \forall v \in V \\
x_{ui} + x_{vi} - w_i &\leq 0 \quad \forall (u, v) \in E \\
x_{vi}, w_i &\leq 1 \quad \forall v \in V, 0 \leq i < n
\end{aligned}$$

Instruction scheduling

- This is an important part of all optimizing compilers.
- The purpose is to minimize delays in the pipeline of a CPU. Assume for simplicity the input is a list of instructions, each with two source operands, one destination operand and a number indicating how many clock cycles it takes a CPU to execute the instruction.
- For instance, for a floating point add instruction, this **latency** may be five clock cycles. The input can be described as a weighted directed acyclic graph with nodes being instructions and with an edge from u to v if instruction u computes the value of an operand of instruction v. The edge the latency of u as weight.
- The instruction scheduling problem is NP-complete for most realistic CPUs. How can we solve it with integer linear programming? Let our goal be to find an m cycle schedule, and that our CPU can issue ("start executing") r instructions each clock cycle.

Instruction scheduling

- Let x_i^j be a decision variable which says instruction i is scheduled in cycle j.
- Our schedule must satisfy the dependences in the dag, i.e., with an edge from u to v, u must not be scheduled after v for correctness, but preferably sufficiently earlier than v to take the latency of u into account.
- Denote by L_{uv} this latency. In this example we look for any feasible solution.
- With the nodes V, with $n = |V|$, and edges E, a basic model then becomes:

Instruction scheduling model

$$\sum_{j=1}^{m} x_i^j = 1 \qquad \forall x_i \in V$$

$$\sum_{i=1}^{n} x_i^j \leq r \qquad \forall x_i 1 \leq j \leq n$$

$$\sum_{j=1}^{m} j \times x_k^j + L_{ki} \leq \sum_{j=1}^{m} j \times x_i^j \qquad \forall (k,i) \in E$$

In the last constraint, the expression $\sum_{j=1}^{m} j \times x_k^j$ is the cycle in which instruction k is scheduled.

EDAG01 Efficient C, LP2

- This course has six labs with the labs 1-4 about implementing an ILP solver in C
- Lab 5 is about understanding what takes time using a simulator from IBM Research
- Lab 6 is about what optimizing compilers can do
- There is a competition and the most recent winner was Filip Raguz (nanoscience)