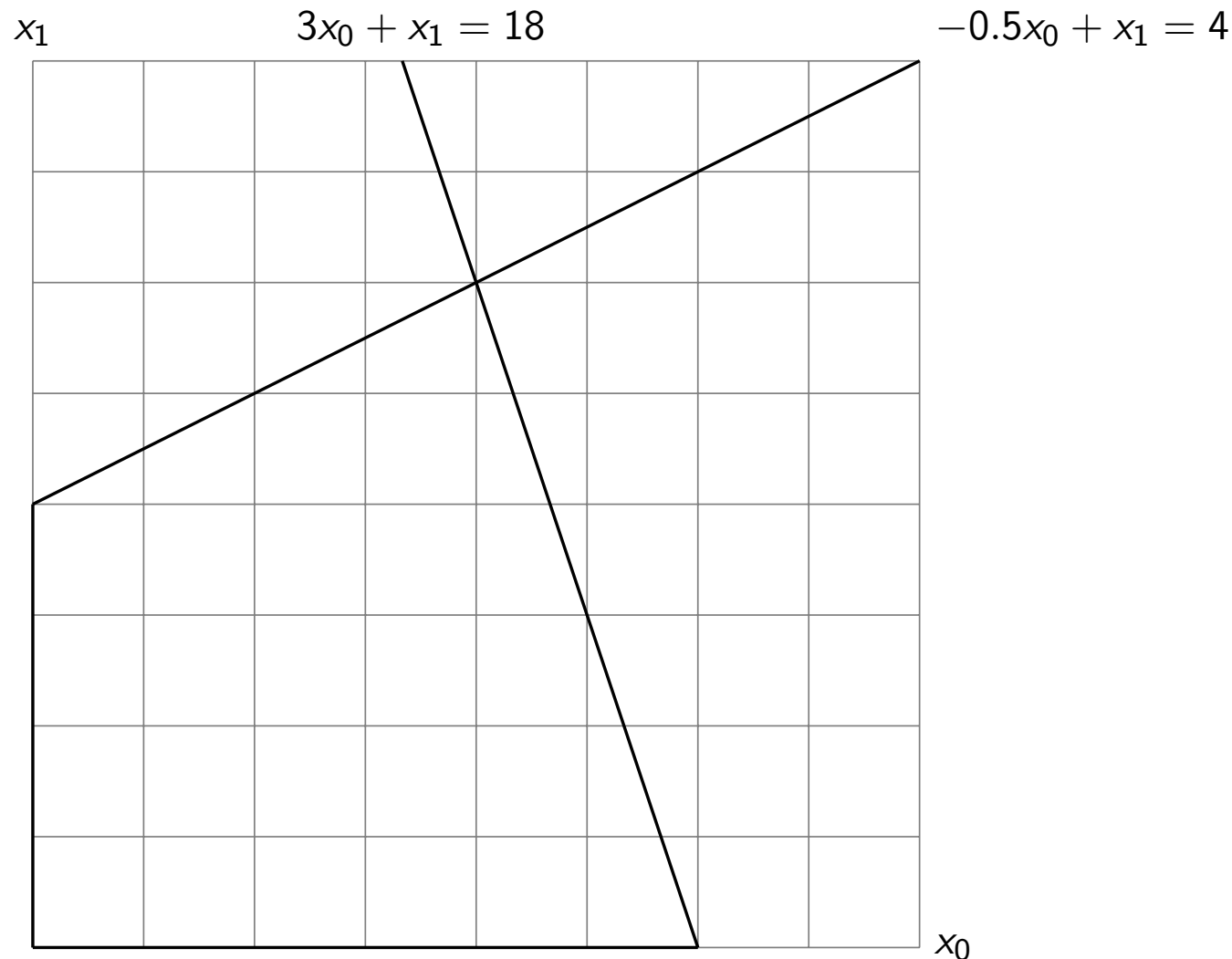


Contents Lecture 2

- Course lab theme and project: integer program solver
- You will implement this during labs 1 — 4
- The labs have other contents as well, such as
 - the gdb debugger
 - Google Sanitizer
 - Valgrind
 - operf, gprof, gcov
- Lab 5: POWER8 pipeline simulator
- Lab 6: optimizing compilers (gcc, clang, and ibm's and nvidia's)

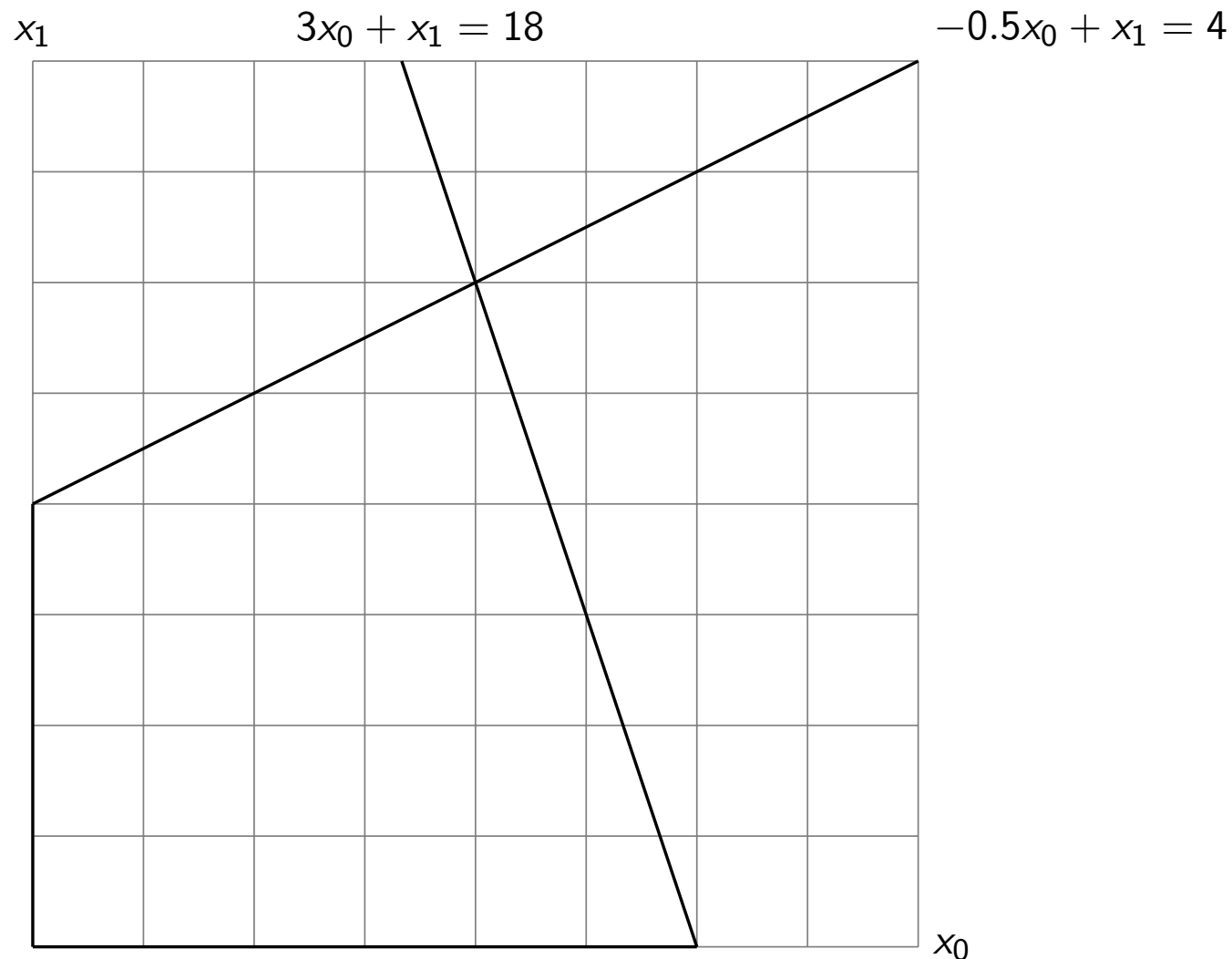
Linear program: maximize a linear function in a region

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



Solving a linear program

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



Our example

- Objective function and linear constraints using \leq

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

$$\begin{array}{rclcl} -0.5x_0 & + & x_1 & \leq & 4 \\ 3x_0 & + & x_1 & \leq & 18. \end{array}$$

- Also: implicitly $x_i \geq 0$
- We can use e.g. $x_i \geq 4$ or $x_i = 5$ but they can be rewritten to use \leq

Linear programs



$$\max \quad z = c_0x_0 + c_1x_1 + \dots + c_{n-1}x_{n-1}$$

$$a_{0,0}x_0 + a_{0,1}x_1 + \dots + a_{0,n-1} \leq b_0$$

$$a_{1,0}x_0 + a_{1,1}x_1 + \dots + a_{1,n-1} \leq b_1$$

...

$$a_{m-1,0}x_0 + a_{m-1,1}x_1 + \dots + a_{m-1,n-1} \leq b_{m-1}$$

$$x_0, x_1, \dots, x_{n-1} \geq 0$$

• or simpler as

$$\max \quad z = cx$$

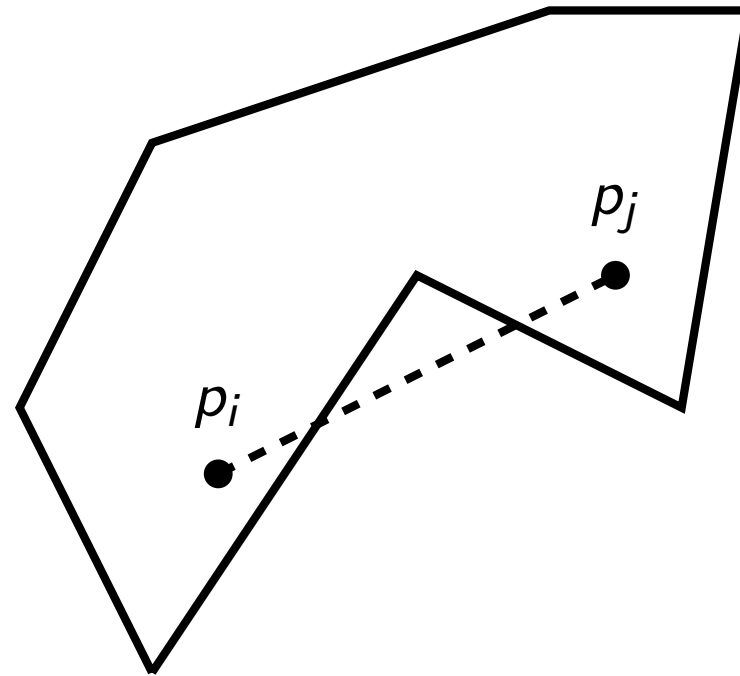
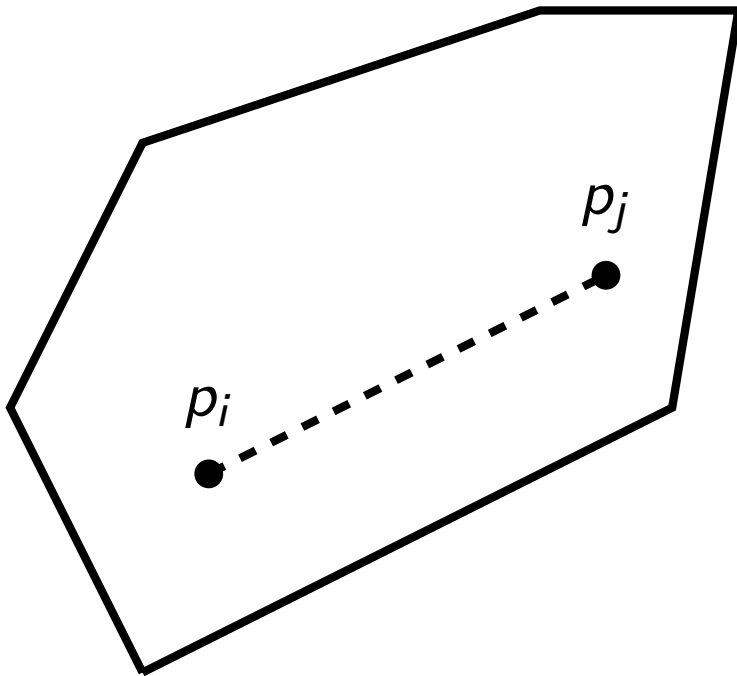
$$Ax \leq b$$

$$x \geq 0.$$

- 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 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.

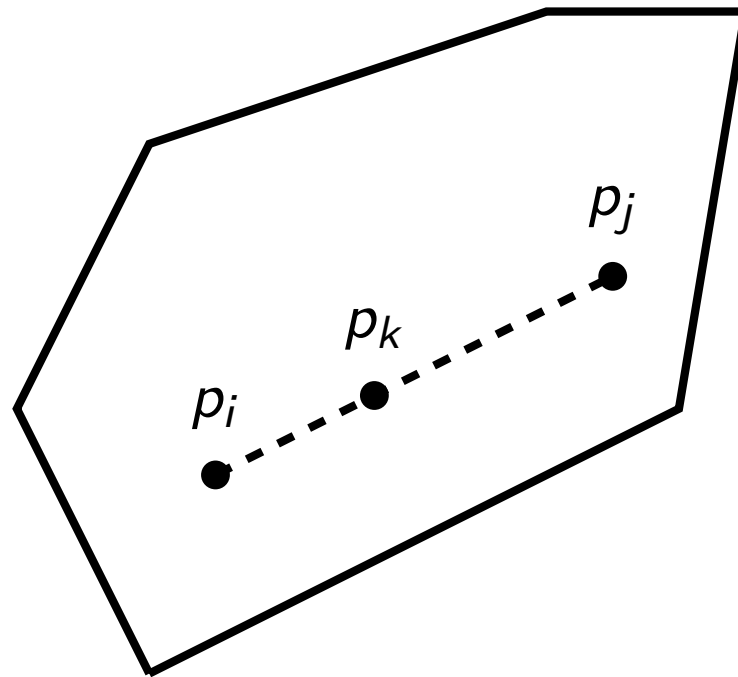
Solutions

- 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 $p_k = \lambda \cdot p_i + (1 - \lambda) \cdot p_j$, with $0 \leq \lambda \leq 1$
- So that p_k is in the region
- Here $\lambda = 0.6$



Local and global optimal solutions

- We denote by $z(x)$ the value of the objective function z at point x .
- A solution x is a **local optimum** for $z(x)$ if there exists an $\epsilon > 0$ such that $z(x) \geq z(y)$ for all $y \in P$ 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 $0 \leq \lambda \leq 1$
- $z(u) \geq z(v) = z(\lambda u + (1 - \lambda)w) = \lambda z(u) + (1 - \lambda)z(w)$
- 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: x^0, x^1, \dots, x^{k-1}
- Let z^* be the maximum value in any vertex: $z^* = \max\{cx^i, 0 \leq i < k\}$
- Every point w in P can be written as a linear combination of the vertices: $w = \sum_{i=0}^{k-1} \lambda_i x^i$ with $\sum_{i=0}^{k-1} \lambda_i = 1$
- Let w be any point in P
- $z(w) = cw = c \sum \lambda_i x^i = \sum \lambda_i (cx^i) \leq \sum \lambda_i z^* = z^* \sum \lambda_i = z^*.$



Slack form

$$\max \quad cX$$

$$x_{n+0} = b_0 - \sum_{j=0}^{n-1} a_{0,j} x_j$$

$$x_{n+1} = b_1 - \sum_{j=0}^{n-1} a_{1,j} x_j \quad (1)$$

...

$$x_{n+m-1} = b_{m-1} - \sum_{j=0}^{n-1} a_{m-1,j} x_j$$

$$x_i \geq 0 \quad 0 \leq i \leq n + m - 1$$

- The variables on the left hand side are called **basic variable** 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

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

$$\begin{array}{rclcl} -0.5x_0 & + & x_1 & \leq & 4 \\ 3x_0 & + & x_1 & \leq & 18 \end{array}$$

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

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

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

- 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**

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

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

- 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

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

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

- 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.

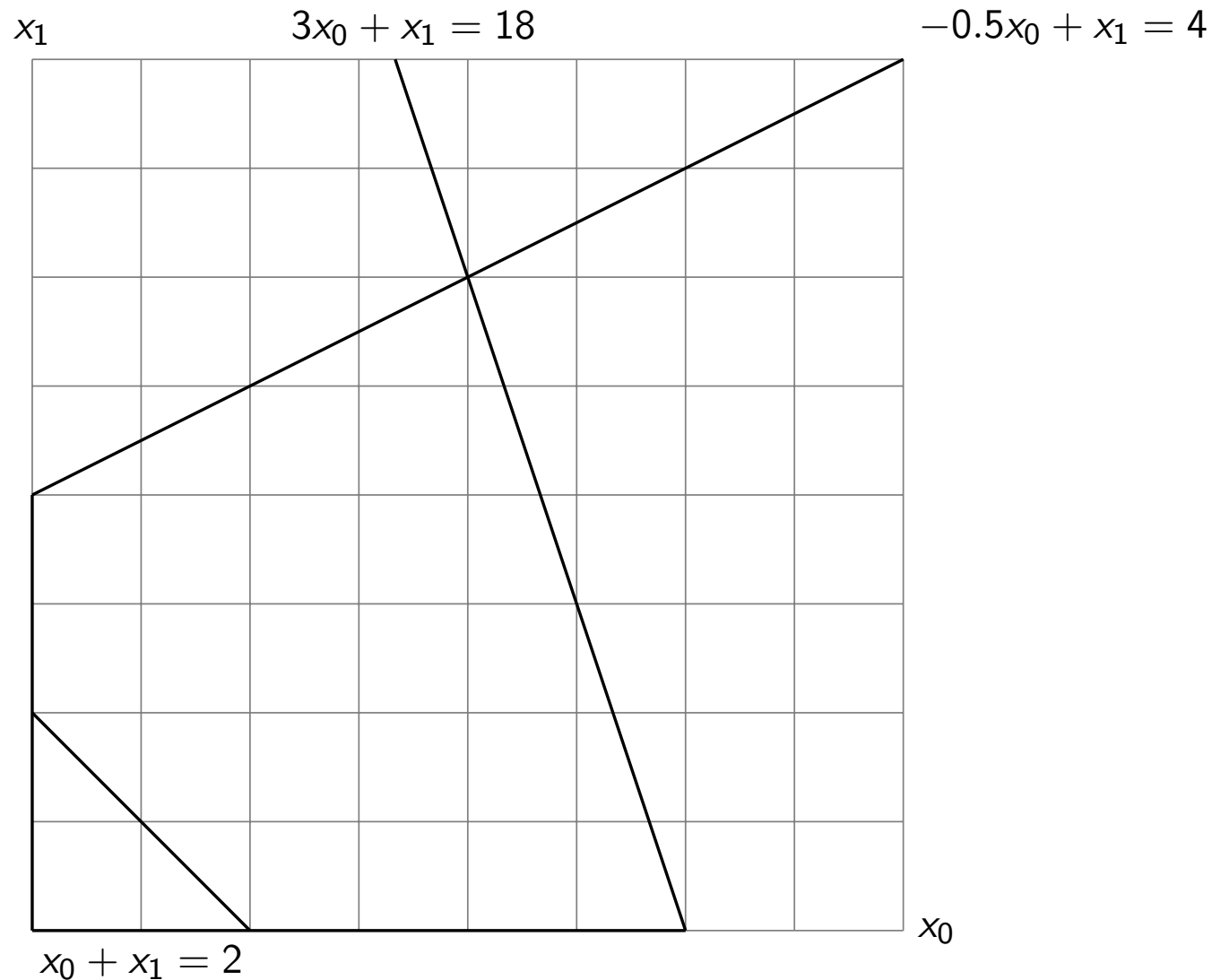
$$\max z = -0.6x_3 - 1.4x_2 + 16$$

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

- 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: $(0, 0)$ not in P

- $x_0 + x_1 = 2$ here means $x_0 + x_1 \geq 2$, i.e. $-x_0 - x_1 \leq -2$, i.e. $b_2 = -2$



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:

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

- 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 $b \geq 0$, 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 condition 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 since integer programming is NP-complete)

- A **relaxation** makes a problem simpler (by solving another problem)
- 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 $x_k \notin \mathbb{N}$
- Assume the Simplex algorithm assigns $x_k = u$
- We can then branch by creating two new linear programs:
 - one with the additional 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 add it in a set h

- 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 set h all unexplored linear programs whose optimal value is less than the value of the integer solution we just found

Pseudo code and project

- The distributed pseudo code is in Appendix B in the book printed 2020 and the course Tresorit directory
- It is as simple possible
- It is your task to translate it to C and then to optimize it.
- The only requirement is that it should work for 20 problems (there are thousands problems).
- Not even commercial programs work perfectly
- At `forsete.cs.lth.se` you will be able to upload your C code
- Forsete is an automatic grader and will give you a score
- The score determines who win a coffee mug and does not affect your grade
- You are **not** allowed to use any code you have not written yourself, except functions from the C Standard library.