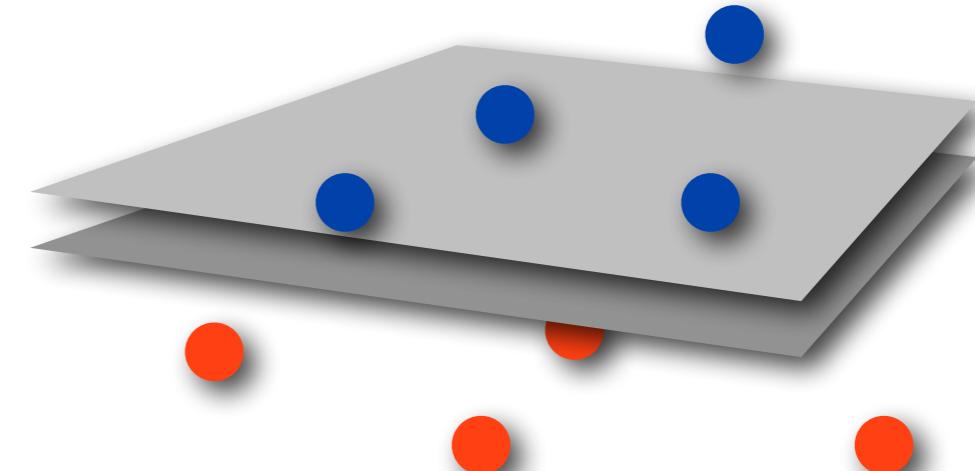
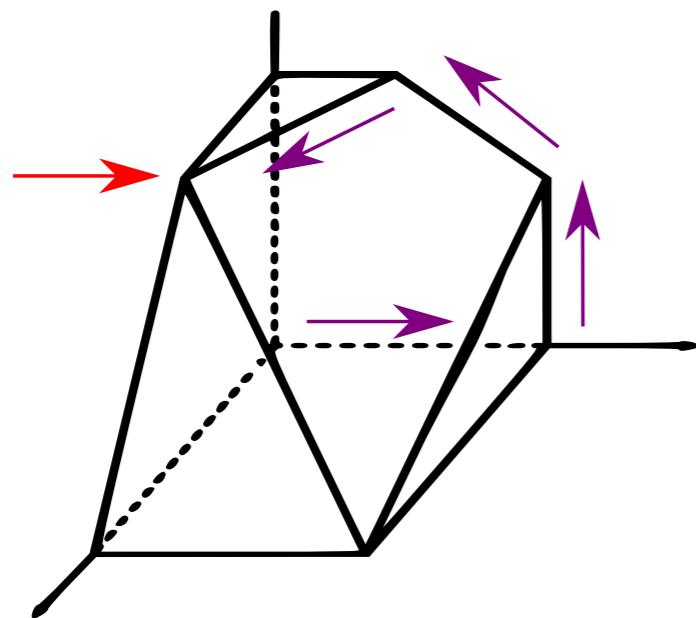


Linear Programming

with



AlgoLab HS21 Tutorial #6 — Hung Hoang

Based on work by Luis Barba, Bernd Gärtner and Sebastian Stich.

Today

- ❖ Quick introduction to Linear Programming
(formulation and representation)
- ❖ Linear Programming in CGAL
- ❖ Examples

Linear Programming (LP)

- ❖ Central topic in optimization
- ❖ Powerful modeling tool in many applications
- ❖ Attracted a lot of attention in optimization during the last six decades for two main reasons:
 - ❖ **Applicability:** Many real-world problems can be modeled using LP;
 - ❖ **Solvability:** Theoretically and practically efficient techniques for solving large-scale problems.

Linear Programming (LP)

An optimization problem subject to constraints.

- ✿ **Variables:** parameters whose values we can choose;
- ✿ **Objective function:** the criterion to minimize (e.g. cost) or maximize (e.g. profit);
- ✿ **Constraints:** limitations for choosing values for the variables.

Obs. Considering minimization is sufficient because minimizing $-f$ is equivalent to maximizing f .

LP—Mathematical Formulation

- ✿ **Problem:** Minimize a linear function in n variables subject to m linear (in)equality constraints.

- ✿ **Example** ($n=2, m=5$):

minimize

$$-32y + 64 \text{ subject to}$$

$$x + y \leq 7$$

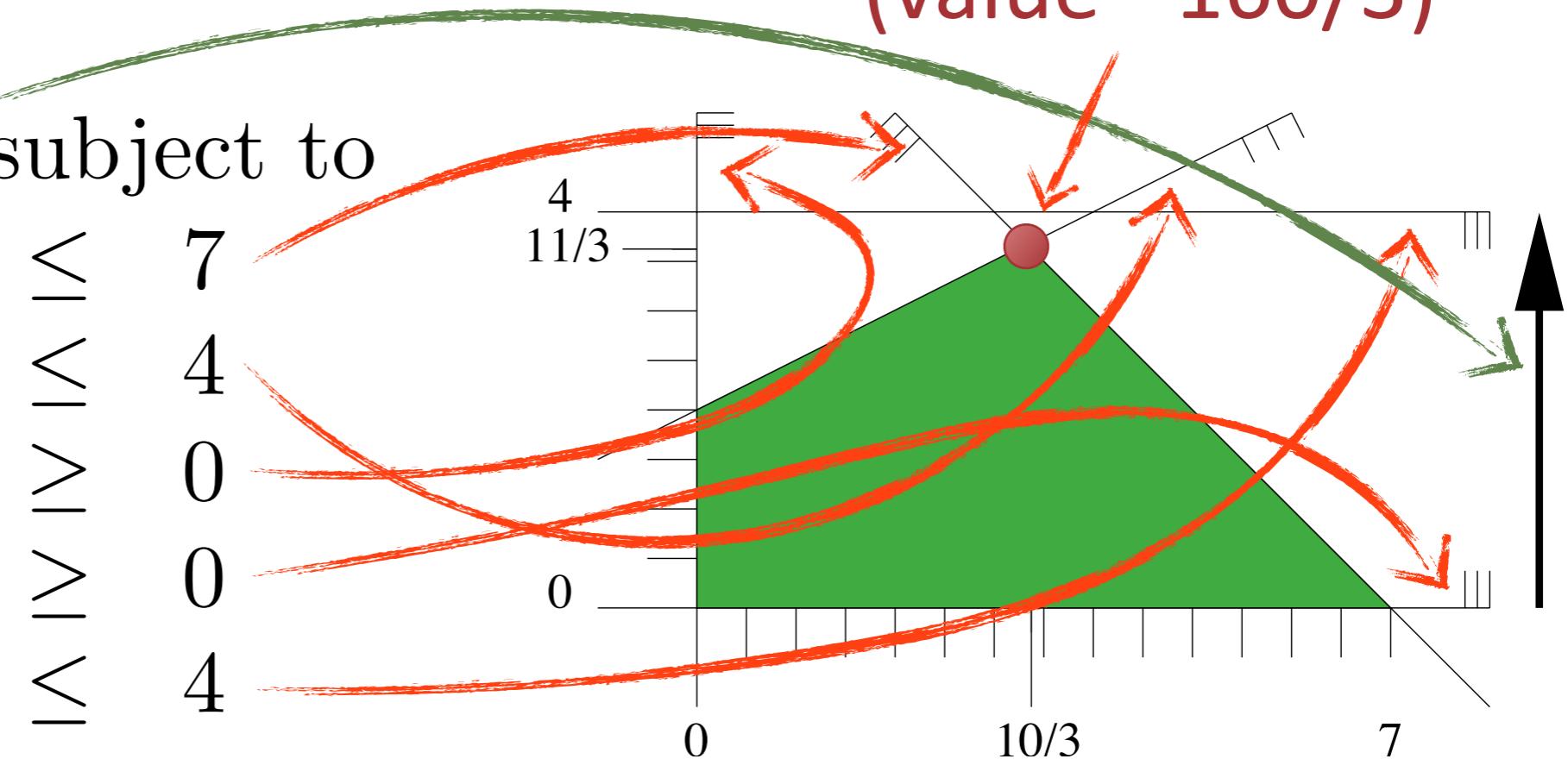
$$-x + 2y \leq 4$$

$$x \geq 0$$

$$y \geq 0$$

$$y \leq 4$$

Optimal solution
(value $-160/3$)



LP—Mathematical Formulation

- ✿ **Problem:** Minimize a linear function in n variables subject to m linear (in)equality constraints.
- ✿ **Example** ($n=2, m=5$):

minimize

$-32y + 64$ subject to

$$x + y \leq 7$$

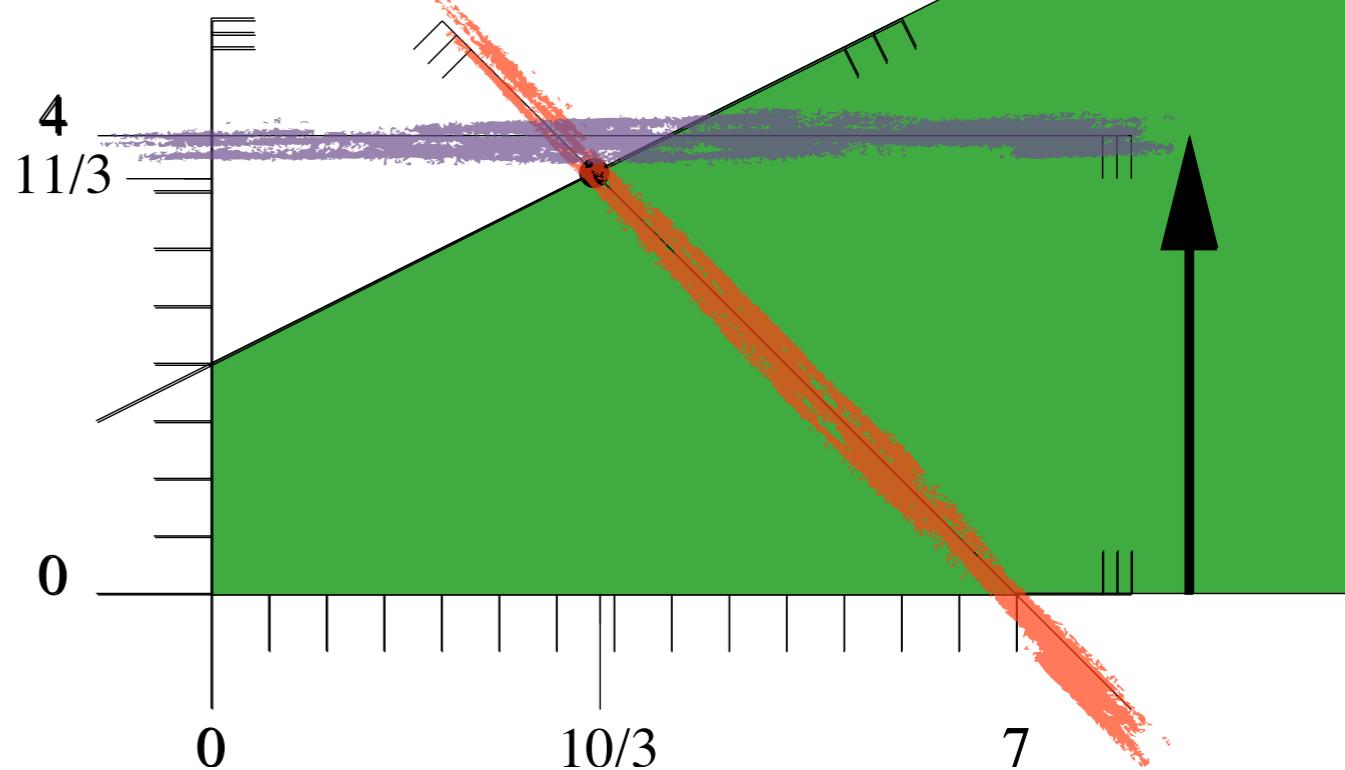
$$-x + 2y \leq 4$$

$$x \geq 0$$

$$y \geq 0$$

$$y \leq 4$$

Unbounded LP



LP—Mathematical Formulation

- ✿ **Problem:** Minimize a linear function in n variables subject to m linear (in)equality constraints.
- ✿ **Example** ($n=2, m=5$):

minimize

$$-32y + 64 \text{ subject to}$$

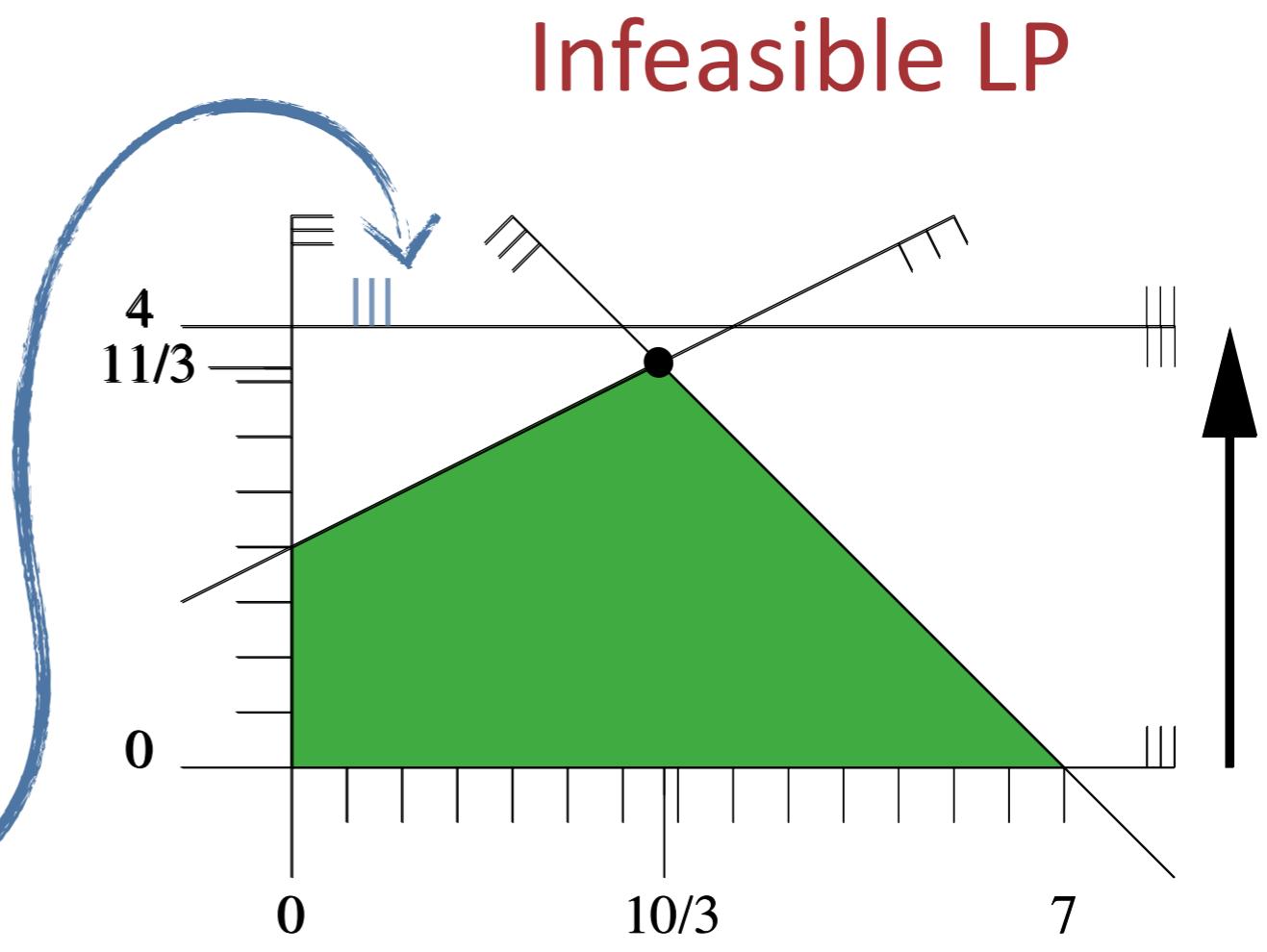
$$x + y \leq 7$$

$$-x + 2y \leq 4$$

$$x \geq 0$$

$$y \geq 0$$

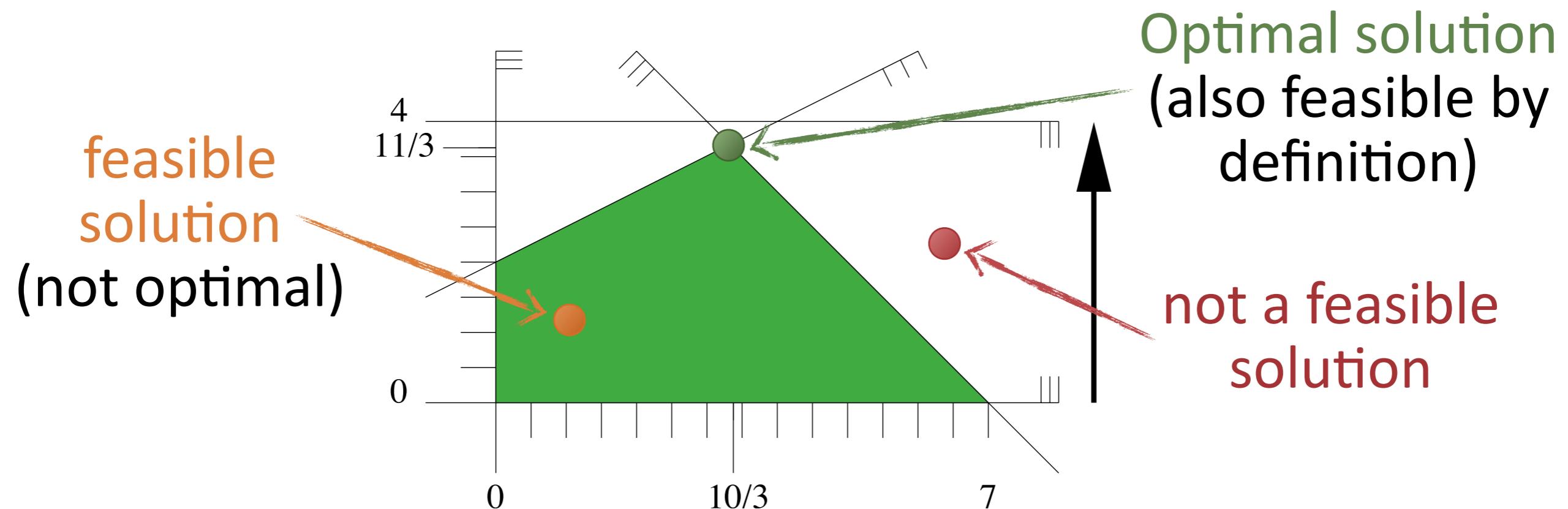
$$y \geq 4$$



Feasible and Optimal Solutions

A **feasible solution** is an assignment of values to the variables of an LP that satisfies all constraints.

A feasible solution is **optimal** if it minimizes the objective function (among all feasible solutions).



Types of Linear Programs

A linear program is of one of the following three types:

- ✿ **Optimal:** there exists an optimal solution (possibly many) that attains the unique minimum of the objective function;
- ✿ **Unbounded:** there exist feasible solutions that attain arbitrarily small values of the objective function;
- ✿ **Infeasible:** There is no feasible solution.

LP—Matrix Formulation

- ✿ **Problem:** Minimize a linear function in n variables subject to m linear (in)equality constraints!

- ✿ **Example** ($n=2, m=5$):

convert “ \geq ” to “ \leq ” by inverting coefficients

minimize

$$-32y + 64 \text{ subject to}$$

$$x + y \leq 7$$

$$-x + 2y \leq 4$$

$$-x \leq 0 \quad x \geq 0$$

$$-y \leq 0 \quad y \geq 0$$

$$y \leq 4$$

A	b
1 1	7
-1 2	4
-1 0	0
0 -1	0
0 1	4

LP—Matrix Formulation

- ✿ **Problem:** Minimize a linear function in n variables subject to m linear (in)equality constraints!

- ✿ **Example** ($n=2, m=5$):

minimize

$-32y + 64$ subject to

$$\begin{array}{lll} x + y & \leq & 7 \\ -x + 2y & \leq & 4 \\ -x & \leq & 0 \\ -y & \leq & 0 \\ y & \leq & 4 \end{array}$$

$$\begin{array}{c} c^T \\ 0 \\ -32 \end{array} \begin{array}{cc} x & y \end{array} + \begin{array}{c} c_0 \\ 64 \end{array} \quad \begin{array}{ccccc} 1 & & 1 & & 7 \\ -1 & & 2 & & 4 \\ -1 & & 0 & & 0 \\ 0 & & -1 & & 0 \\ 0 & & 1 & & 4 \end{array} \quad \begin{array}{c} A \\ x \\ y \\ b \end{array}$$

LP—General Form in CGAL

$$\begin{array}{ll} \text{minimize} & c^T x + c_0 \\ \text{subject to} & Ax \stackrel{\leq, =, \geq}{\gtrless} b \\ & \ell \leq x \leq u \end{array}$$

$\leq, =, \geq$ (individually
for each constraint)

variables $(x, c, \ell, u \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c_0 \in \mathbb{R})$

objective function

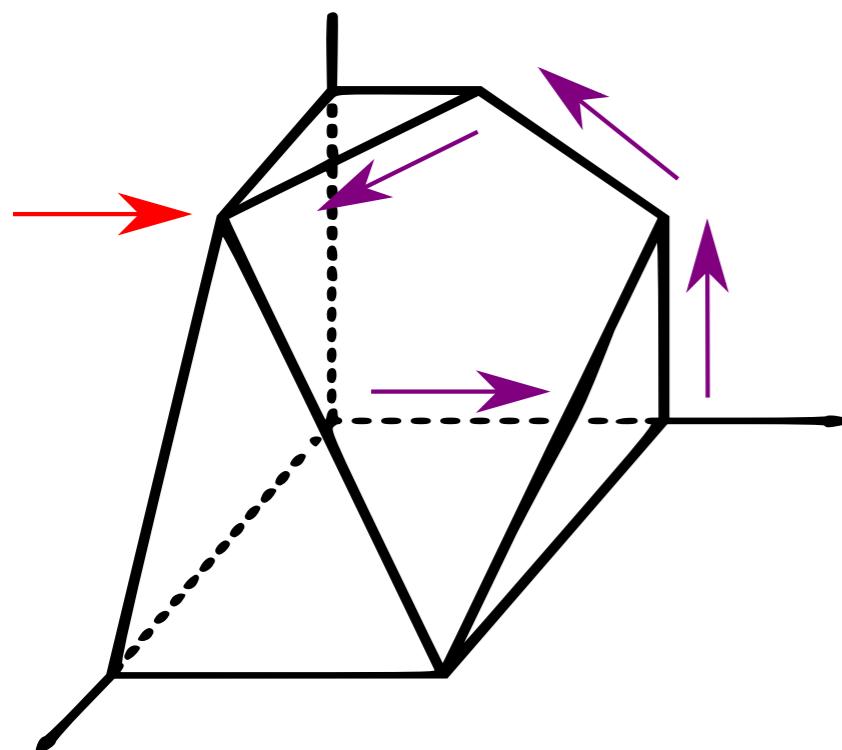
lower and upper bounds

constraint matrix

right-hand side

shift

LP—Geometric Interpretation

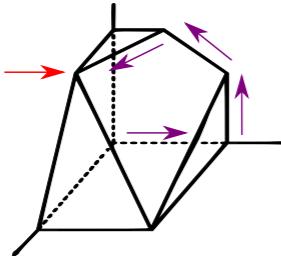


$$\begin{array}{ll}\text{minimize} & c^T x + c_0 \\ \text{subject to} & Ax \geq b \\ & \ell \leq x \leq u\end{array}$$

$$P = \{x | Ax \leq b\}$$

- The constraints define an n -dimensional convex polyhedron P with $\leq m$ faces.
- Optimum objective value is attained at a vertex of P .
- But $\#\text{vertices}(P)$ can be exponential in m and n .
=> Trying all vertices not feasible.

LP—Complexity of CGAL Solver



$$\begin{aligned} & \text{minimize} && c^T x + c_0 \\ & \text{subject to} && Ax \geq b \\ & && \ell \leq x \leq u \\ & && P = \{x | Ax \leq b\} \end{aligned}$$

- The constraints define an n -dimensional convex polyhedron P with $\leq m$ faces.
- CGAL solver uses a simplex-type algorithm: Walk from vertex to vertex along edges of P while improving the objective value.
- Worst-case complexity is exponential in n and m .

For $\min \{n, m\}$ small, the complexity is $O(\max \{n, m\})$.



LP in CGAL—Preamble

- ✿ Choose input type (in this order of preference):
 - ✿ `int, long`
 - ✿ `CGAL::Gmpz` (arbitrary precision integer)
 - ✿ `CGAL::Gmpq` (arbitrary precision rational)
- ✿ Choose exact type for solver:
 - ✿ `CGAL::Gmpq` (if this is the input type)
 - ✿ `CGAL::Gmpz` (in all other cases)

```
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

// choose input type (input coefficients must fit)
typedef int IT;
// choose exact type for solver (CGAL::Gmpz or CGAL::Gmpq)
typedef CGAL::Gmpz ET;
```

LP in CGAL—Preamble (2)

- ❖ Define program and solution type:

```
// program and solution types
typedef CGAL::Quadratic_program<IT> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;
```

- ❖ Yep, it is called Quadratic... (more general concept) but also used for linear programs.
- ❖ Internally, the solver uses [CGAL::Quotient<ET>](#).

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{lcl} x + y & \leq & 7 \\ -x + 2y & \leq & 4 \\ x & \geq & 0 \\ y & \geq & 0 \\ y & \leq & 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

LP in CGAL—Setup

- Enter the program data:

Default inequality is \leq .

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);
```

```
// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

$$\begin{aligned} \text{minimize} \quad & -32y + 64 = c^T x + c_0 \\ \text{subject to} \quad & \left\{ \begin{array}{lll} Ax \leq b & \left\{ \begin{array}{lll} x + y & \leq & 7 \\ -x + 2y & \leq & 4 \\ x & \geq & 0 \\ y & \geq & 0 \\ y & \leq & 4 \end{array} \right. \end{array} \right. \end{aligned}$$

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);
// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

Lower bounds: All variables must be \geq zero.

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

No upper bounds.

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);
```

```
// set the coefficients of A and b
const int X = 0;
const int Y = 1;
```

```
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

Convenient alias
(index) for variables.

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);
```

```
// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

Constraint/inequality #0.

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);
```

```
// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

Constraint/inequality #1.

LP in CGAL—Setup

- Enter the program data:

You can use `set_a()` and `set_b()` for the upper bound, too. But we recommend `set_u()` for convenience and efficiency.

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{l} x + y \leq 7 \\ -x + 2y \leq 4 \\ x \geq 0 \\ y \geq 0 \\ y \leq 4 \end{array} \right. \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

Upper bound for y.

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{lcl} x + y & \leq & 7 \\ -x + 2y & \leq & 4 \\ x & \geq & 0 \\ y & \geq & 0 \\ y & \leq & 4 \end{array} \right. \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

LP in CGAL—Setup

- Enter the program data:

$$\begin{aligned} & \text{minimize} && -32y + 64 = c^T x + c_0 \\ & \text{subject to} && \left\{ \begin{array}{lcl} x + y & \leq & 7 \\ -x + 2y & \leq & 4 \\ x & \geq & 0 \\ y & \geq & 0 \\ y & \leq & 4 \end{array} \right. \\ & Ax \leq b && \end{aligned}$$

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

// set the coefficients of A and b
const int X = 0;
const int Y = 1;
lp.set_a(X, 0, 1); lp.set_a(Y, 0, 1); lp.set_b(0, 7); // x + y <= 7
lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4); // -x + 2y <= 4
// set upper bound
lp.set_u(Y, true, 4); // y <= 4
// objective function
lp.set_c(Y, -32); // -32y
lp.set_c0(64); // +64
```

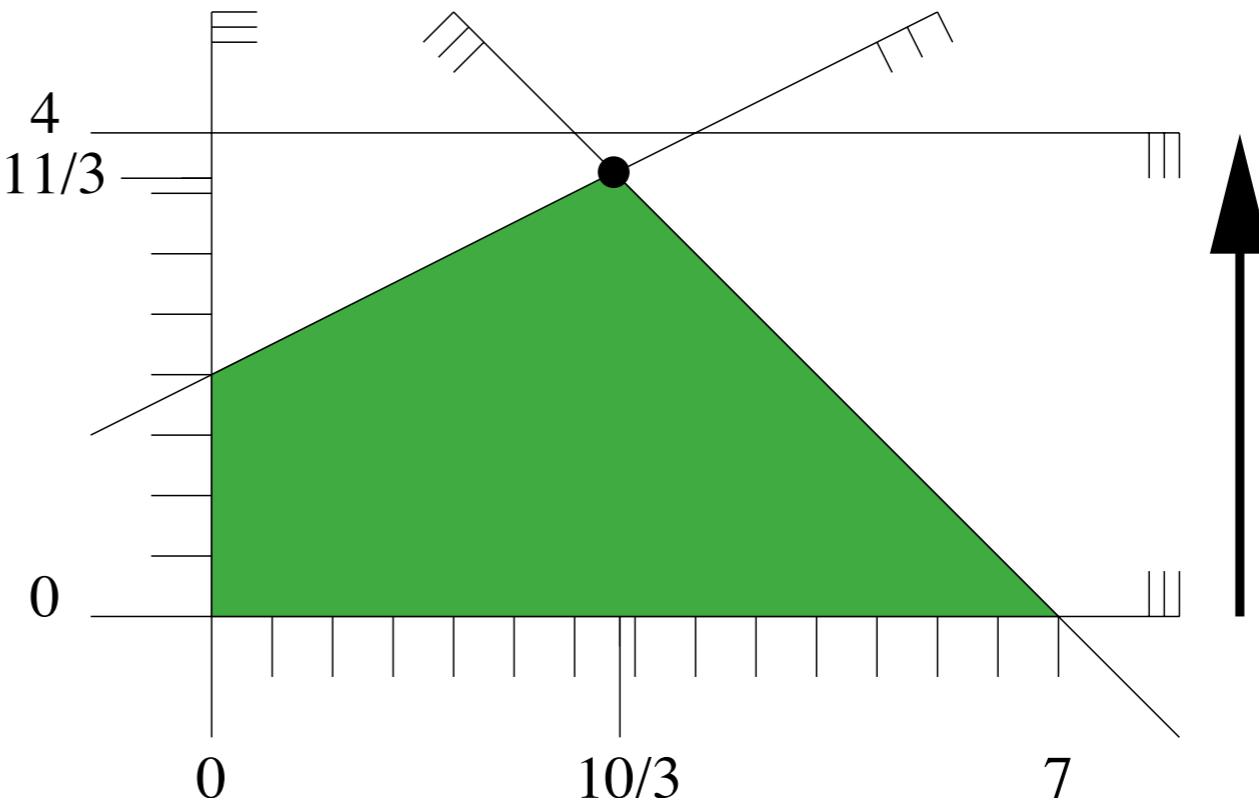
LP in CGAL—Solve

- Call the LP solver and output the solution:

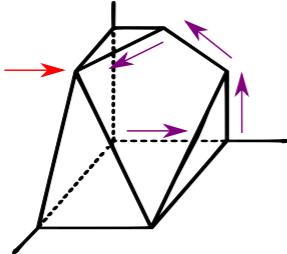
```
// solve the program, using ET as the exact type
Solution s = CGAL::solve_linear_program(lp, ET());
// output solution
std::cout << s;
```

- Output:

```
status:          OPTIMAL
objective value: -160/3
variable values:
 0: 10/3
 1: 11/3
```



When to apply LP?

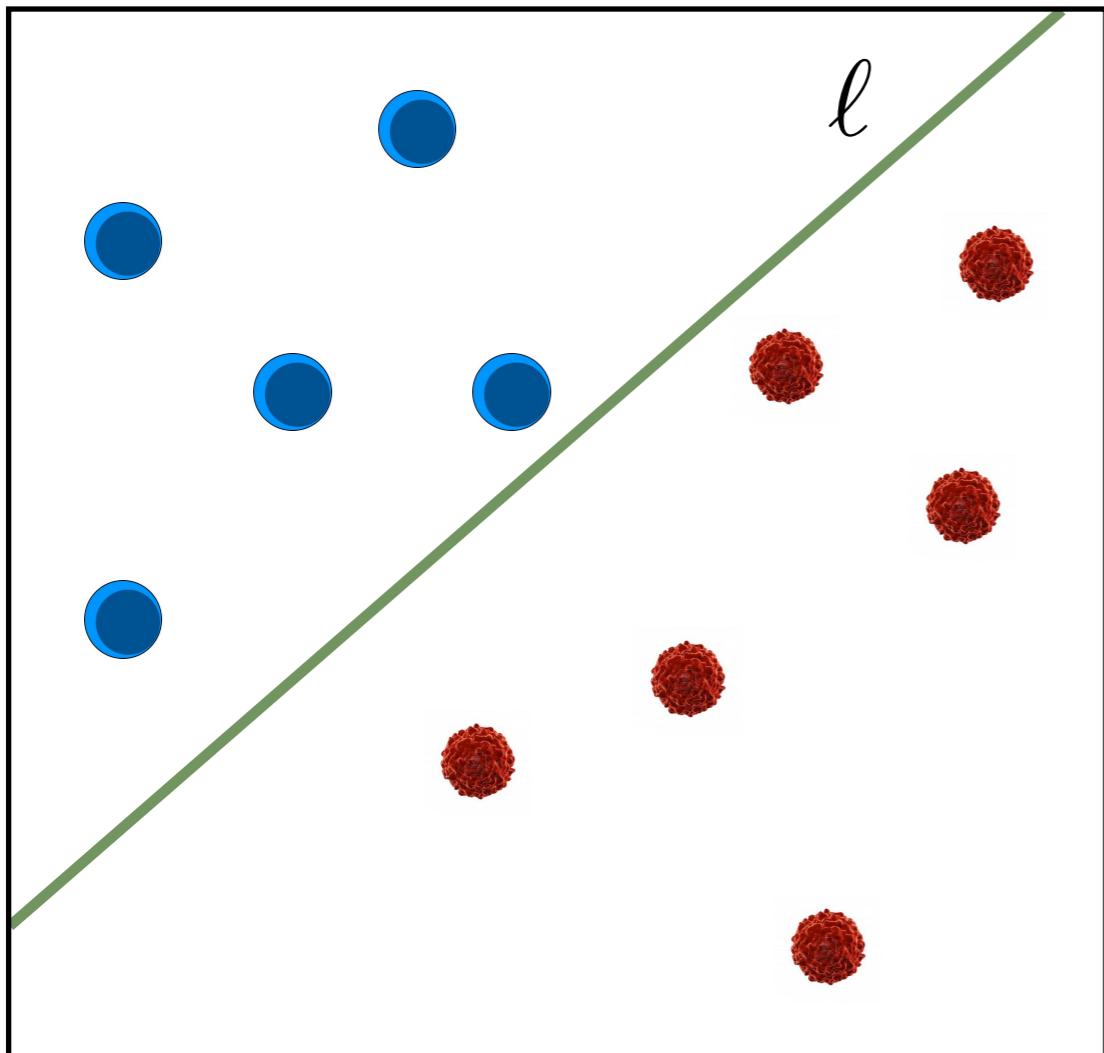


$$\begin{aligned} & \text{minimize} && c^T x + c_0 \\ & \text{subject to} && Ax \geq b \\ & && \ell \leq x \leq u \\ & && P = \{x | Ax \leq b\} \end{aligned}$$

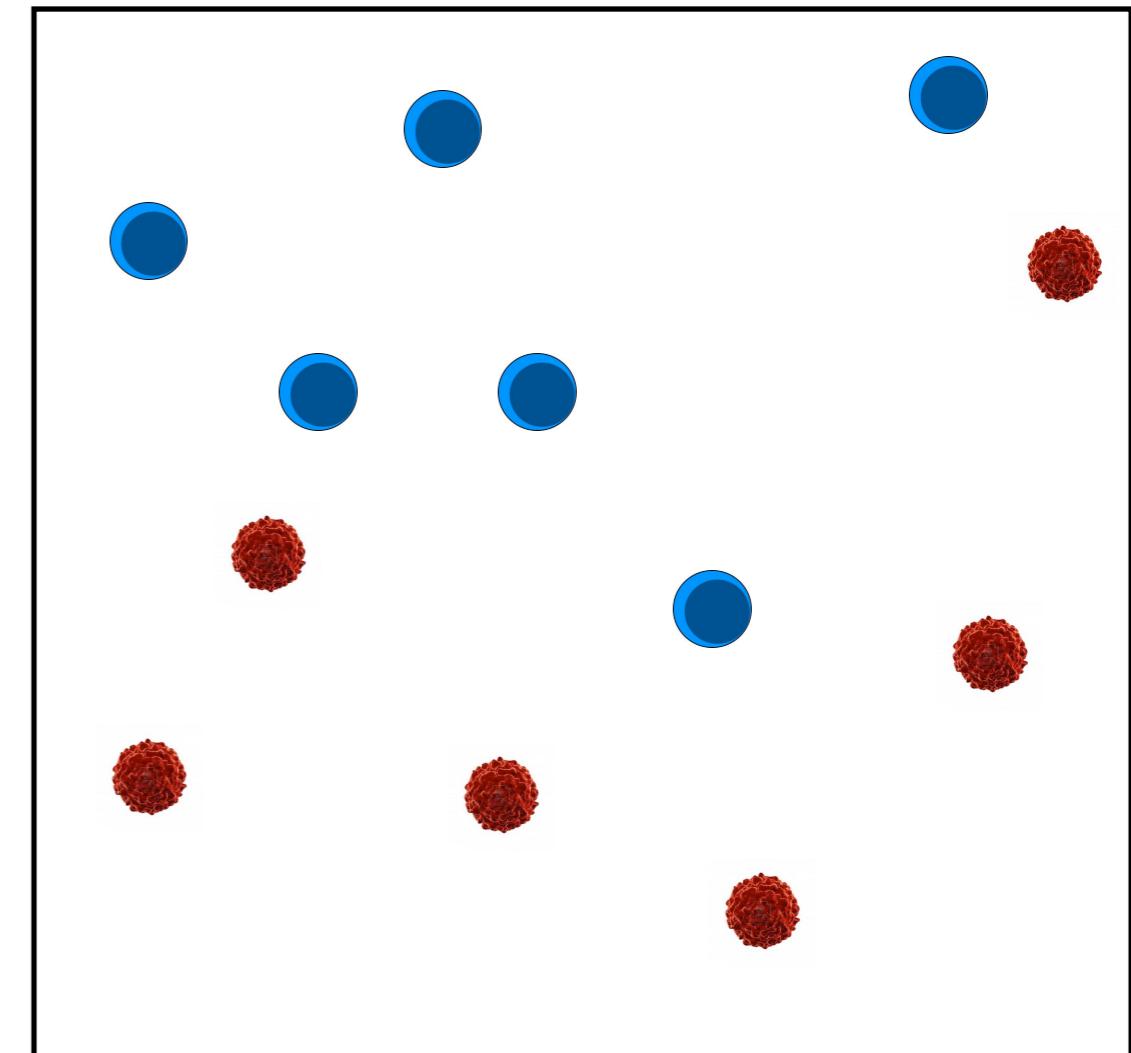
- ✿ If a problem can be modelled using n variables and m constraints s.t.
 - ✿ constraints are **linear** (in)equalities in the variables
 - ✿ at least **one of n or m is small.**
- ✿ Agenda/Checklist:
 - ✿ What are **variables**, what are **constraints**? Output
 - ✿ What is n , what is m ? Is one of n or m small? Input
 - ✿ Are all constraints linear in the variables?

Ex. I: Linear Separation

- Given: A set R of red points and a set B of blue points.
- Want: A line ℓ that separates R and B .



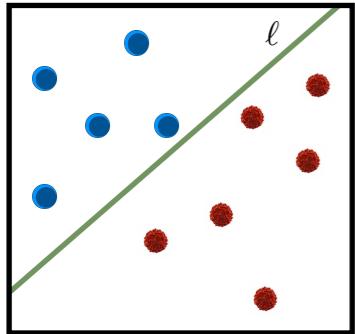
Linearly separable



No solution 😞

Ex. I: Linear Separation

- Given: A set R of red points and a set B of blue points.
- Want: A line ℓ that separates R and B .



- Agenda/Checklist:
- What are variables, what are constraints?
- What is n , what is m ? Is one of n or m small?
- Are all constraints linear in the variables?

$n=3 \Rightarrow$ small
 $m = |R| + |B|$

Yes!

- Variables/Output: We are looking for a line ℓ .
 $\ell : \{(x,y) : ax+by+c=0\}$, for a,b,c unknown.
- Constraints/Input: R and B on different sides of ℓ .
 $ap_x + bp_y \leq c$, for $(p_x, p_y) \in R$ and
 $aq_x + bq_y \geq c$, for $(q_x, q_y) \in B$.

Linear Separation

```
// example: decide whether two point sets R and B can be
// separated by a line

#include <iostream>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

// choose input type (input coefficients must fit)
typedef int IT;
// choose exact type for solver (CGAL::Gmpz or CGAL::Gmpq)
typedef CGAL::Gmpz ET;

// program and solution types
typedef CGAL::Quadratic_program<IT> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main()
{
    // create an LP with Ax <= b and no lower/upper bounds
    Program lp(CGAL::SMALLER, false, 0, false, 0);
    const int a = 0;
    const int b = 1;
    const int c = 2;

    // number of red and blue points
    int m; std::cin >> m;
    int n; std::cin >> n;
}
```

```
// read the red points
for (int i = 0; i < m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c <= 0
    lp.set_a(a, i, x);
    lp.set_a(b, i, y);
    lp.set_a(c, i, 1);
}
// read the blue points
for (int i = 0; i < n; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c >= 0
    lp.set_a(a, m+i, -x);
    lp.set_a(b, m+i, -y);
    lp.set_a(c, m+i, -1);
}
// no objective function, just feasibility
// solve the program, using ET as the exact type
Solution s = CGAL::solve_linear_program(lp, ET());
assert(s.solves_linear_program(lp));

// output solution
std::cout << s;
}
```

Output for
 $R=\{(0,0)\}$ and
 $B=\{(1,1)\}$:

status: OPTIMAL
objective value: 0/1
variable values:
0: 0/1
1: 0/1
2: 0/1

Oops, not a line...

Linear Separation

```
// example: decide whether two point sets R and B can be
// separated by a nonvertical line

#include <iostream>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

// choose input type (input coefficients must fit)
typedef int IT;
// choose exact type for solver (CGAL::Gmpz or CGAL::Gmpq)
typedef CGAL::Gmpz ET;

// program and solution types
typedef CGAL::Quadratic_program<IT> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main()
{
    // create an LP with Ax <= b and no lower/upper bounds
    Program lp(CGAL::SMALLER, false, 0, false, 0);
    const int a = 0;
    const int b = 1;
    const int c = 2;

    // number of red and blue points
    int m; std::cin >> m;
    int n; std::cin >> n;
}
```

```
// read the red points
for (int i = 0; i < m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c <= 0
    lp.set_a(a, i, x);
    lp.set_a(b, i, y);
    lp.set_a(c, i, 1);
}
// read the blue points
for (int i = 0; i < n; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c >= 0
    lp.set_a(a, m+i, -x);
    lp.set_a(b, m+i, -y);
    lp.set_a(c, m+i, -1);
}
// enforce that b is positive
lp.set_l(b, true, 1);

// solve the program, using ET as the exact type
Solution s = CGAL::solve_linear_program(lp, ET());
assert(s.solves_linear_program(lp));

// output solution
std::cout << s;
}
```

Output for
 $R=\{(0,0)\}$ and
 $B=\{(1,1)\}$:

status: OPTIMAL
objective value: 0/1
variable values:
0: 0/1
1: 1/1
2: 0/1

Fix: Set $b \neq 0$.

But this is not a valid inequality.

Linear Separation

```
// example: decide whether two point sets R and B can be
// separated by a nonvertical line

#include <iostream>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

// choose input type (input coefficients must fit)
typedef int IT;
// choose exact type for solver (CGAL::Gmpz or CGAL::Gmpq)
typedef CGAL::Gmpz ET;

// program and solution types
typedef CGAL::Quadratic_program<IT> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main()
{
    // create an LP with Ax <= b and no lower/upper bounds
    Program lp(CGAL::SMALLER, false, 0, false, 0);
    const int a = 0;
    const int b = 1;
    const int c = 2;

    // number of red and blue points
    int m; std::cin >> m;
    int n; std::cin >> n;
}
```

```
// read the red points
for (int i = 0; i < m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c <= 0
    lp.set_a(a, i, x);
    lp.set_a(b, i, y);
    lp.set_a(c, i, 1);
}
// read the blue points
for (int i = 0; i < n; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up constraint a x + b y + c >= 0
    lp.set_a(a, m+i, -x);
    lp.set_a(b, m+i, -y);
    lp.set_a(c, m+i, -1);
}
// enforce that b is positive
lp.set_l(b, true, 1);

// solve the program, using ET as the exact type
Solution s = CGAL::solve_linear_program(lp, ET());
assert(s.solves_linear_program(lp));

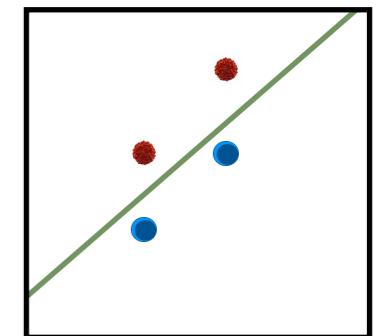
// output solution
std::cout << s;
}
```

Output for
 $R=\{(0,1), (1,2)\}$ and
 $B=\{(0,0), (1,1)\}$:

status: INFEASIBLE

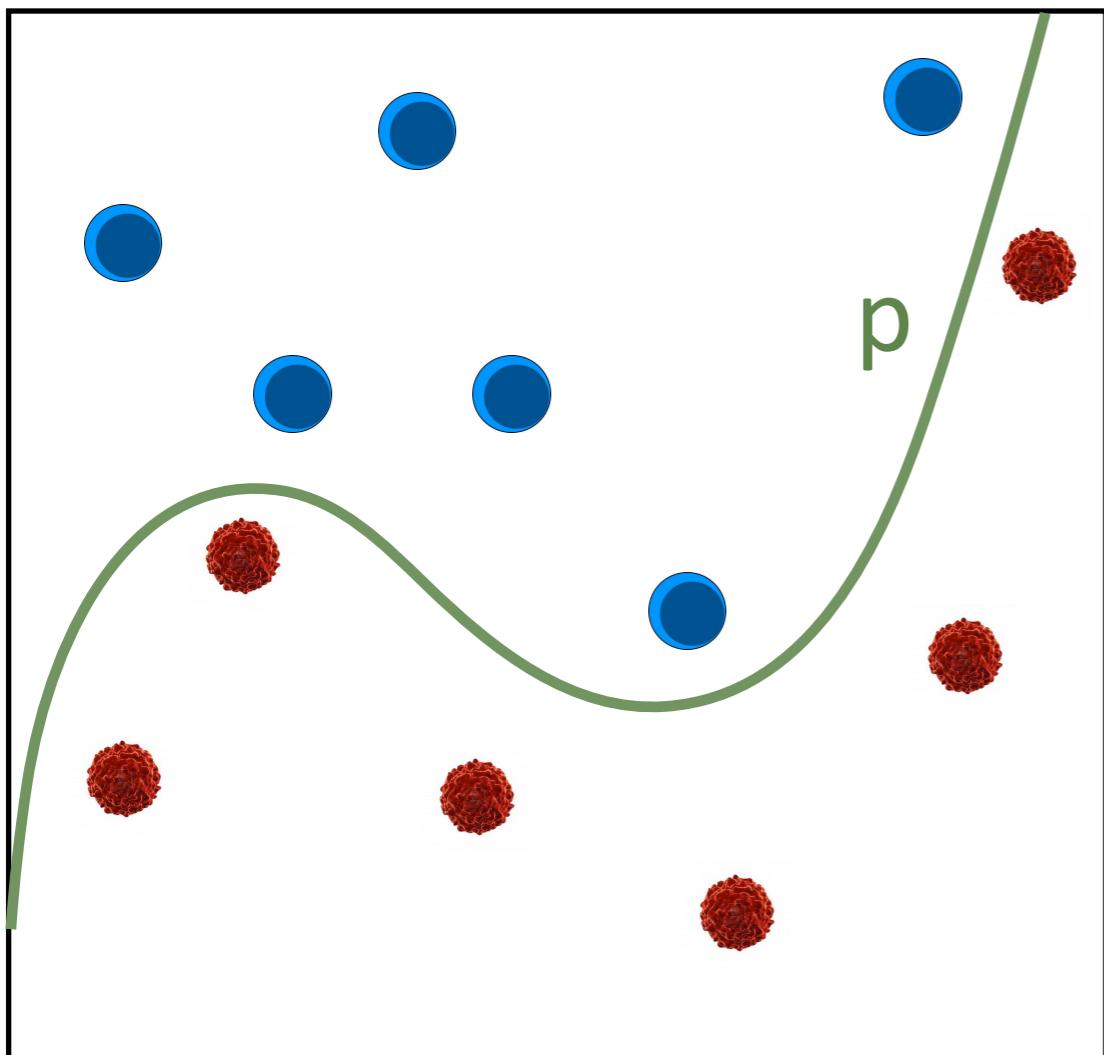
But R and B are linearly separable...

Fix: If the LP is infeasible, change the constraint $b \geq 1$ into $b \leq -1$ and re-solve.

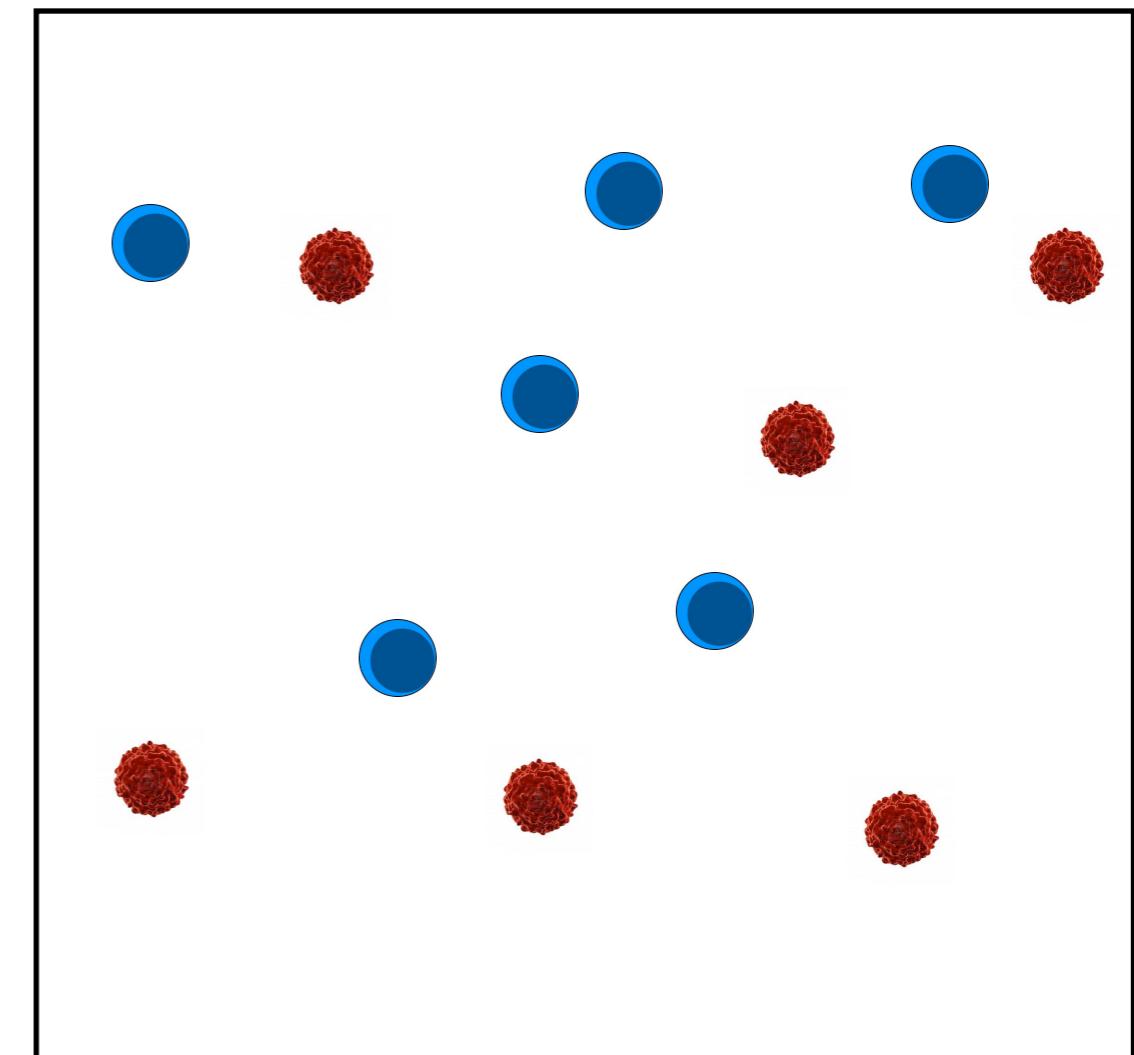


Ex. 2: Cubic Separation

- Given: A set R of red points and a set B of blue points.
- Want: A cubic polynomial p that separates R and B .



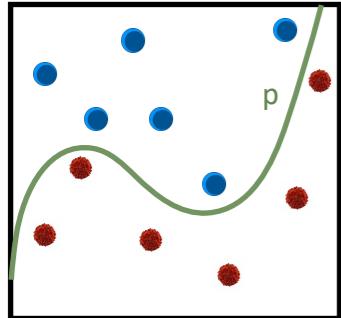
Separable



No solution 😞

Ex. 2: Cubic Separation

- Given: A set R of red points and a set B of blue points.
- Want: A cubic polynomial p that separates R and B .



Agenda/Checklist:

- What are variables, what are constraints?
- What is n , what is m ? Is one of n or m small?
- Are all constraints linear in the variables?

$n=10 \Rightarrow$ small
 $m = |R| + |B|$

Yes!

- Variables/Output: We want a cubic polynomial p :

$$\{(x,y) : ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j\},$$

for a, \dots, j unknown.

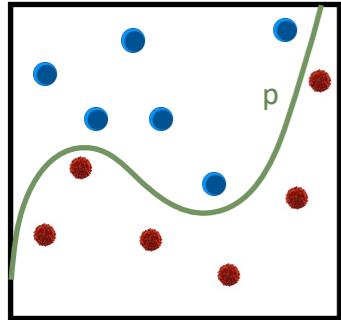
- Constraints/Input: R and B on different sides of p .

$$ap_x^3 + bp_x^2p_y + cp_xp_y^2 + dp_y^3 + ep_x^2 + fp_xp_y + gp_y^2 + hp_x + ip_y \leq j, \forall (p_x, p_y) \in R,$$

$$aq_x^3 + bq_x^2q_y + cq_xq_y^2 + dq_y^3 + eq_x^2 + fq_xq_y + gq_y^2 + hq_x + iq_y \geq j, \forall (q_x, q_y) \in B.$$

Ex. 2: Cubic Separation

- Given: A set R of red points and a set B of blue points.
- Want: A cubic polynomial p that separates R and B .



- Agenda/Checklist:
- What are variables, what are constraints?
- What is n , what is m ? Is one of n or m small?
- Are all constraints linear in the variables?

$$n=10 \Rightarrow \text{small}$$
$$m = |R| + |B|$$

→ Why? There are powers here...

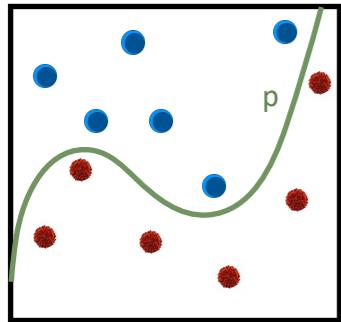
- Variables: $a, b, c, d, e, f, g, h, i, j$.
- Constraints: R and B on different sides of p .

$$ap_x^3 + bp_x^2p_y + cp_xp_y^2 + dp_y^3 + ep_x^2 + fp_xp_y + gp_y^2 + hp_x + ip_y \leq j, \forall (p_x, p_y) \in R,$$
$$aq_x^3 + bq_x^2q_y + cq_xq_y^2 + dq_y^3 + eq_x^2 + fq_xq_y + gq_y^2 + hq_x + iq_y \geq j, \forall (q_x, q_y) \in B.$$

- Constraints are linear in $a, b, c, d, e, f, g, h, i, j$.

Ex. 2: Cubic Separation

- Given: A set R of red points and a set B of blue points.
- Want: A cubic polynomial p that separates R and B .



- Agenda/Checklist:
- What are variables, what are constraints?
- What is n , what is m ? Is one of n or m small?
- Are all constraints linear in the variables?

→ Why? There are powers here...

- Ex. For $(2,3) \in R$ the constraint

$$ap_x^3 + bp_x^2p_y + cp_xp_y^2 + dp_y^3 + ep_x^2 + fp_xp_y + gp_y^2 + hp_x + ip_y \leq j$$

reads

$$8a + 12b + 18c + 27d + 4e + 6f + 9g + 2h + 3i \leq j.$$

Careful with Indices

- The matrix A is adjusted dynamically to the entries and indices you provide.

```
// create an LP with Ax <= b, lower bound 0 and no upper bounds
Program lp (CGAL::SMALLER, true, 0, false, 0);

const int X = 10000;
lp.set_a(X, 0, 1);
// now A has >= 10000 columns...
```

- Even if all these coefficients are zero, this affects efficiency (and space usage).

Nonnegative Solver

- There is a special solver for the case where all variables are nonnegative.

```
// solve the program, using ET as the exact type  
Solution s = CGAL::solve_nonnegative_linear_program(lp, ET());
```

- It may (or may not) be more efficient than the general solver on specific instances.
- **Attention!** The nonnegative solver ignores all other lower and upper bounds (specified with the constructor or with `set_l` or `set_u`).

Processing Solutions

- ❖ There are various functions to access the solution.

```
// solve the program, using ET as the exact type
Solution s = CGAL::solve_linear_program(lp, ET());
if (s.is_optimal()) { ... }
if (s.objective_value() < 0) { ... }
if (s.is_unbounded()) { ... }
if (s.is_infeasible()) { ... }
Solution::Variable_value_iterator opt = s.variable_values_begin();
// keep in mind that the values are of type Quotient<ET>
CGAL::Quotient<ET> res = *opt;
std::cout << res.numerator() << "/" << res.denominator() << "\n";
```

- ❖ In a maximization problem you probably want to invert the objective value...

How to avoid cycling

- ⌘ The solver is driven by a pricing strategy.
- ⌘ By default this strategy is deterministic and **may cycle** => run into an infinite loop.
- ⌘ To avoid cycling, use Bland's Rule (see code below).

```
CGAL::Quadratic_program_options options;  
options.set_pricing_strategy(CGAL::QP_BLAND);  
Solution s = CGAL::solve_linear_program(lp, ETC), options);
```

- ⌘ Bland's Rule is slower => only use where needed.
- ⌘ Usually you will notice on the public test sets.

Known Bug :=(

- ✿ Do not copy or assign objects of type
CGAL::Quadratic_program_solution<ET>!
- ✿ **Workaround 1:** If you want to pass or return such an instance to/from a function, pass a pointer to the instance instead!
- ✿ **Workaround 2:** Just don't do it!