Linear and Quadratic Programming (with CGAL)

Luis Barba, Algorithms Lab

November 1, 2017

Based on slides by Bernd Gärtner

Today

- Quick introduction to Linear Programming (formulation and representation).
- Quick recap on techniques to solve it (no extensive knowledge needed).
- Linear Programming in CGAL.
- Examples.
- Integer Programming (not required for this class, but important).
- Quadratic Convex Programming (CGAL).

Linear Programming (LP)

- Linear programming is a central topic in optimization .
- It provides a powerful tool in modeling many applications .
- LP has attracted most of its attention in optimization during the last six decades for two main reasons :
    - Applicability : There are many real-world applications that can be modeled as linear programming ;
    - Solvability : There are theoretically and practically efficient techniques for solving large-scale problems.

Linear Programming (LP)

We face an optimization problem subject to some constraints.

- **Variables:** describe our choices, the parameters that we are allowed to change;
- **Objective function:** Describes the criterion that we wish to minimize (e.g. cost) or maximize (e.g. profit);
- **Constraints:** Describe the limitations that we have for the choice of the values of the variables.

Linear Programming

Mathematical formulation

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

Linear Programming

Mathematical formulation

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

- **Example** (n=2, m=5)**:**

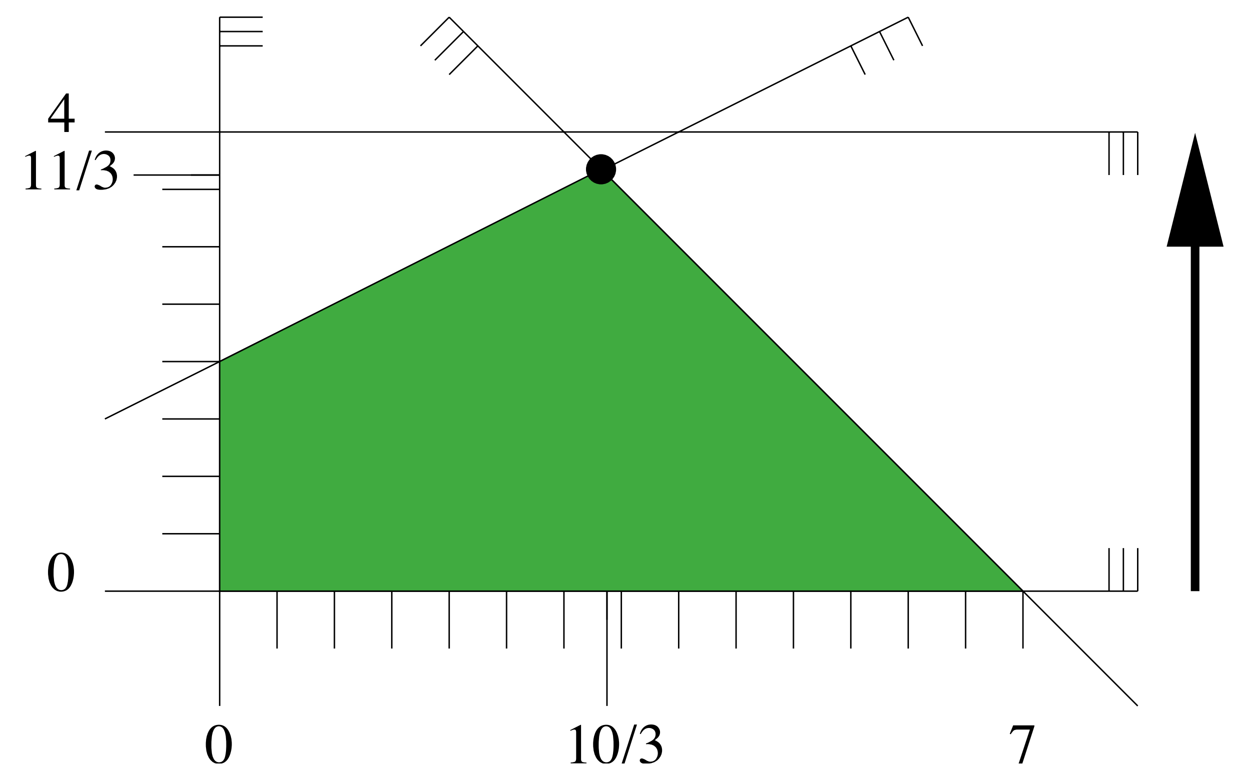
$$
\begin{array}{lrcl}
\text{minimize} & -32y+64 & & \\
\text{subject to} & x+y & \leq & 7 \\
& -x+2y & \leq & 4 \\
& x & \geq & 0 \\
& y & \geq & 0 \\
& y & \leq & 4
\end{array}
$$

Linear Programming

Mathematical formulation

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

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
& -x + 2y & \leq & 4 \\
& x & \geq & 0 \\
& y & \geq & 0 \\
& y & \leq & 4
\end{array}
$$



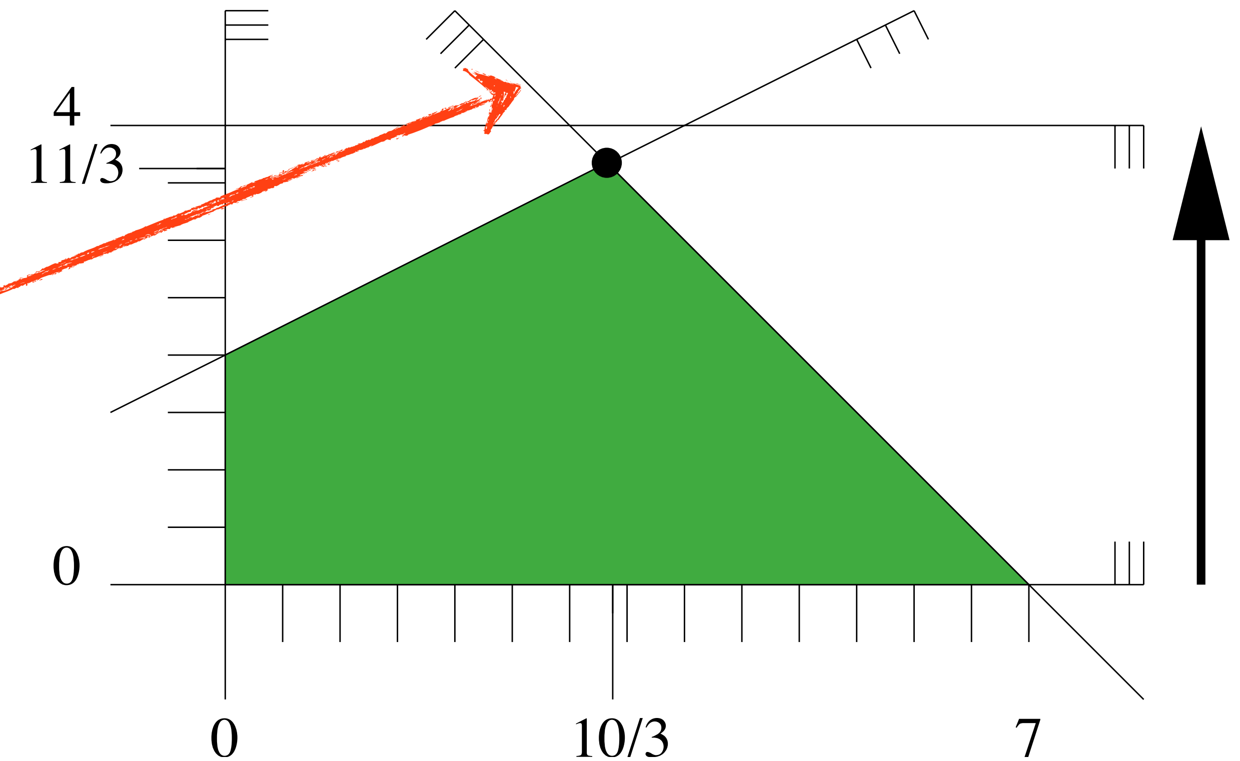
Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$
\begin{array}{llcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
$$



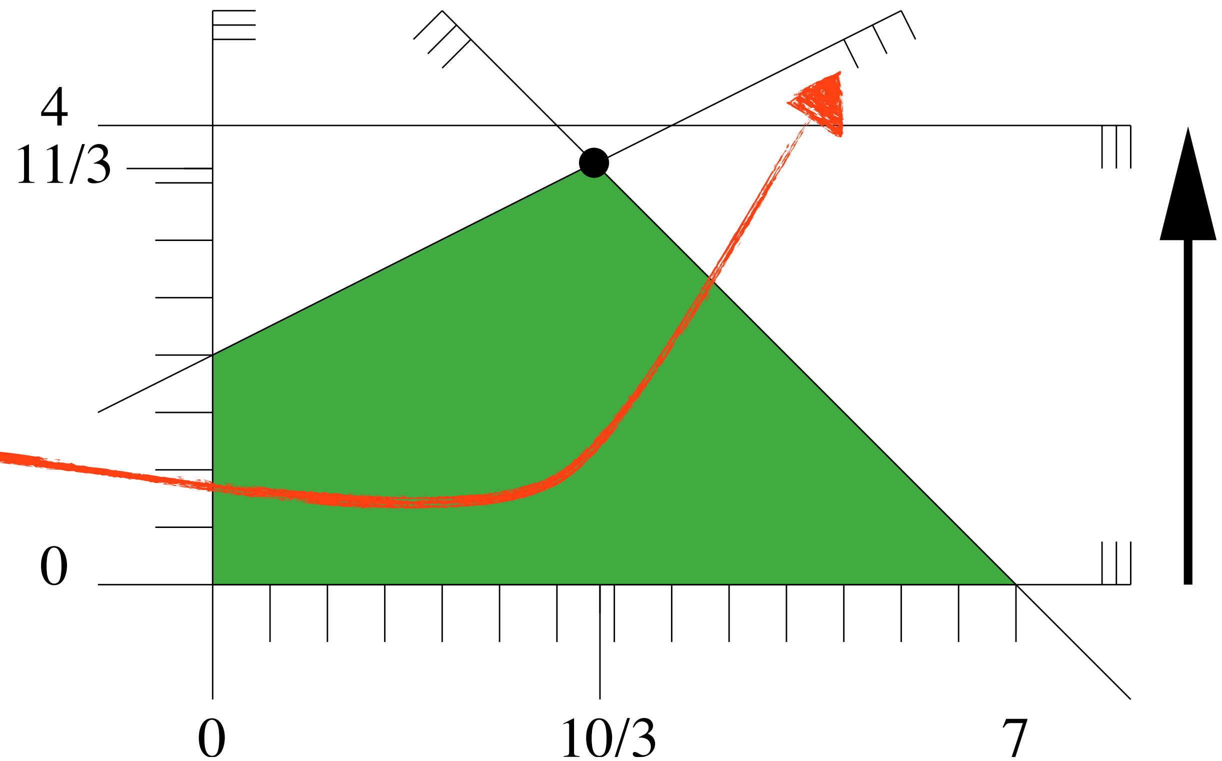
Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5):

$$\begin{array}{lrcl} \text{minimize} & -32y + 64 & & \\ \text{subject to} & x + y & \leq & 7 \\ & -x + 2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array}$$



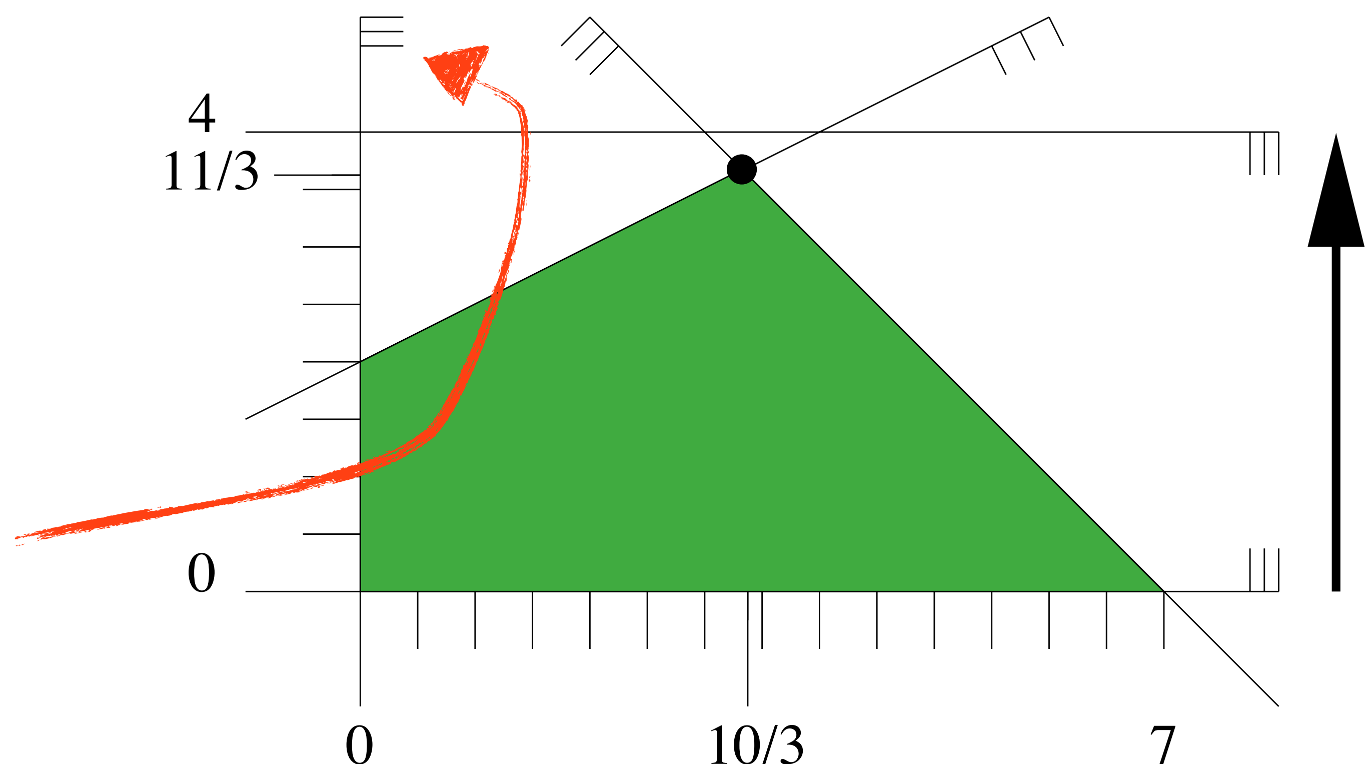
Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$\begin{array}{lrcl} \text{minimize} & -32y+64 & & \\ \text{subject to} & x+y & \leq & 7 \\ & -x+2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array}$$



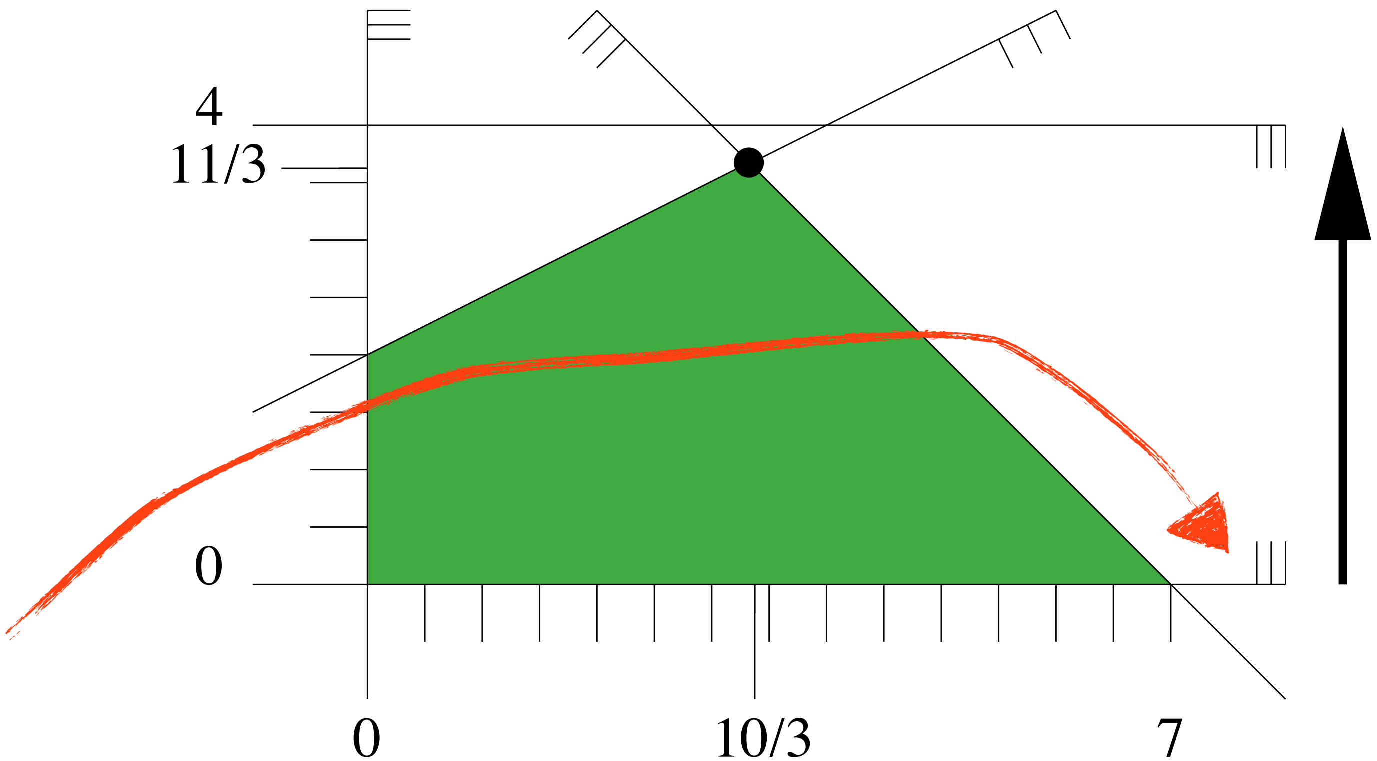
Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$\begin{array}{lrcl} \text{minimize} & -32y + 64 & & \\ \text{subject to} & x + y & \leq & 7 \\ & -x + 2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array}$$

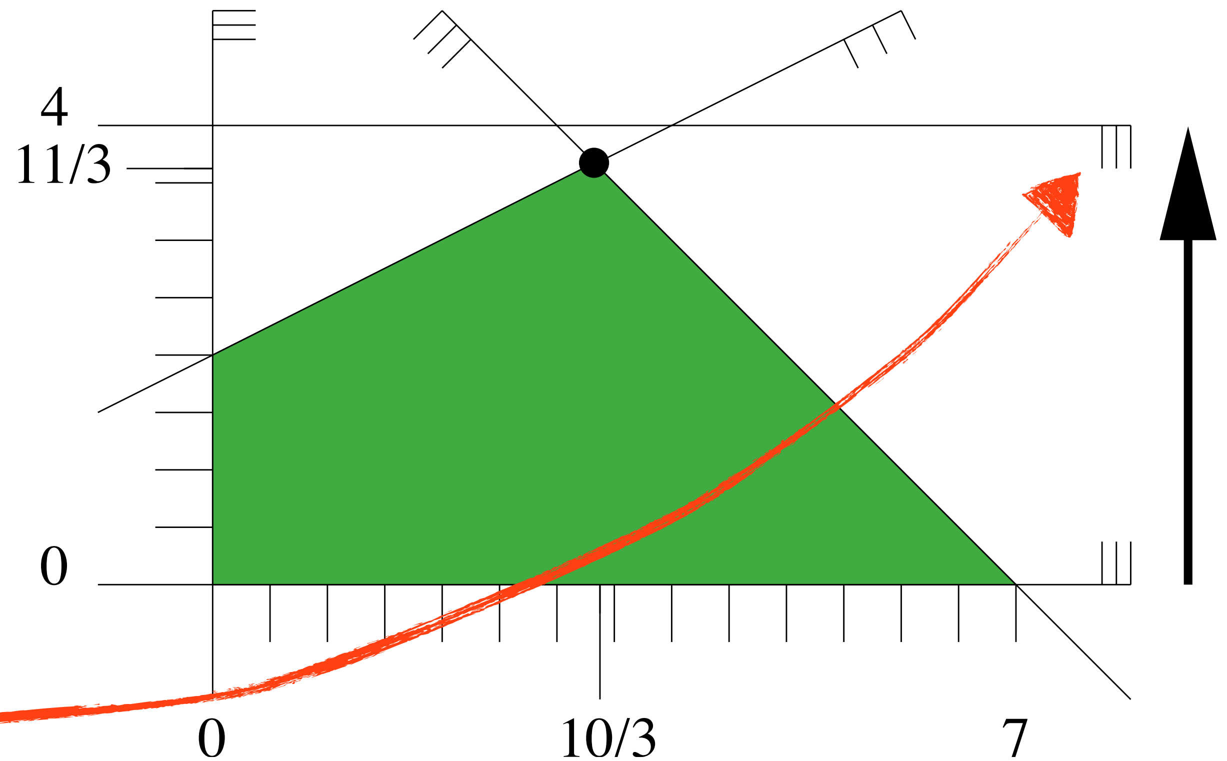


Linear Programming

Mathematical formulation

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

$$
\begin{array}{llcl}
\text{minimize} & -32y+64 & & \\
\text{subject to} & x+y & \leq & 7 \\
 & -x+2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
$$



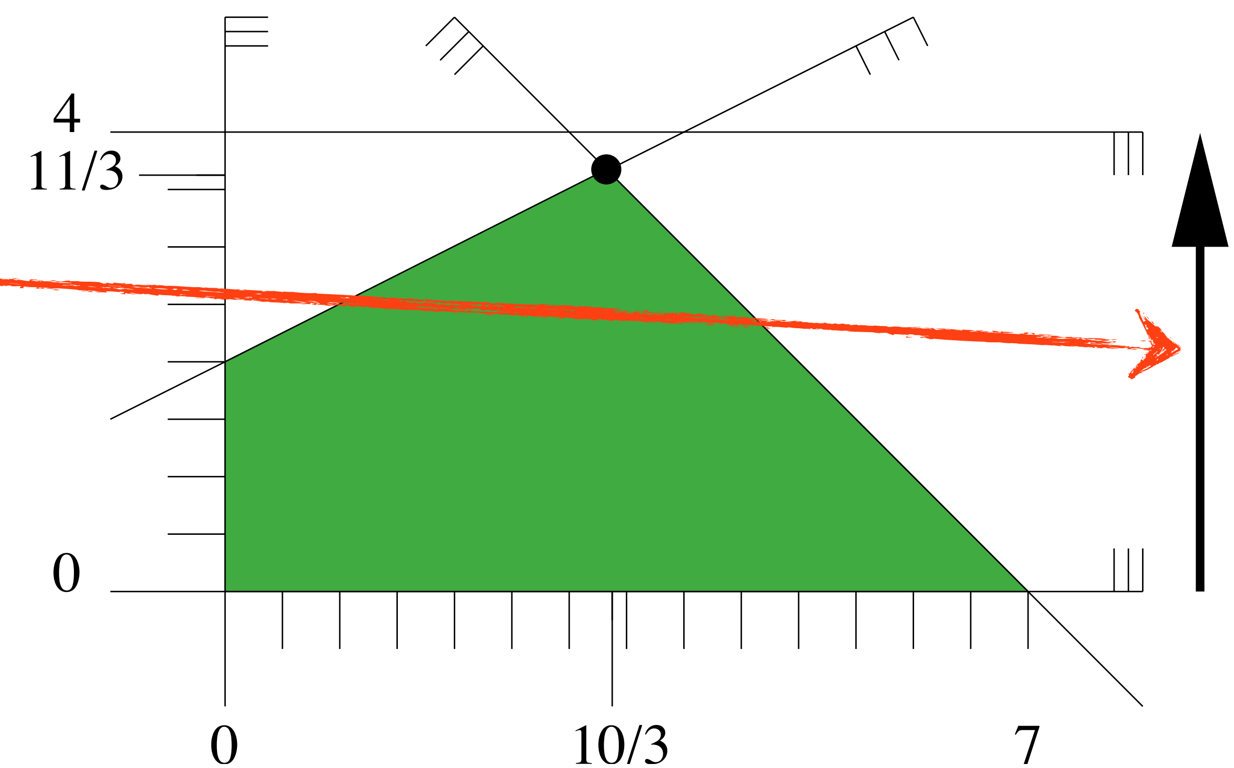
Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$\begin{array}{lrcl} \text{minimize} & -32y+64 & & \\ \text{subject to} & x+y & \leq & 7 \\ & -x+2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array}$$



Linear Programming

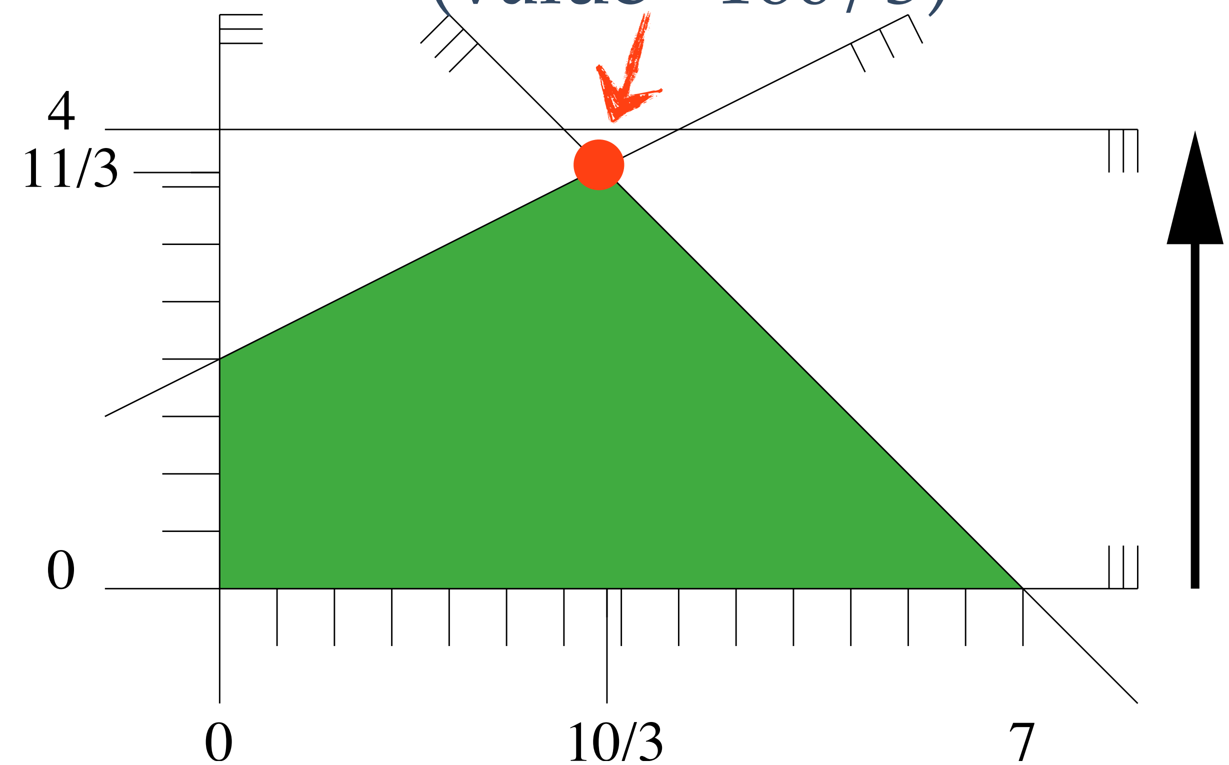
Mathematical formulation

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

- **Example** (*n*=2, *m*=5):

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
$$

Optimal solution
(value -160/3)



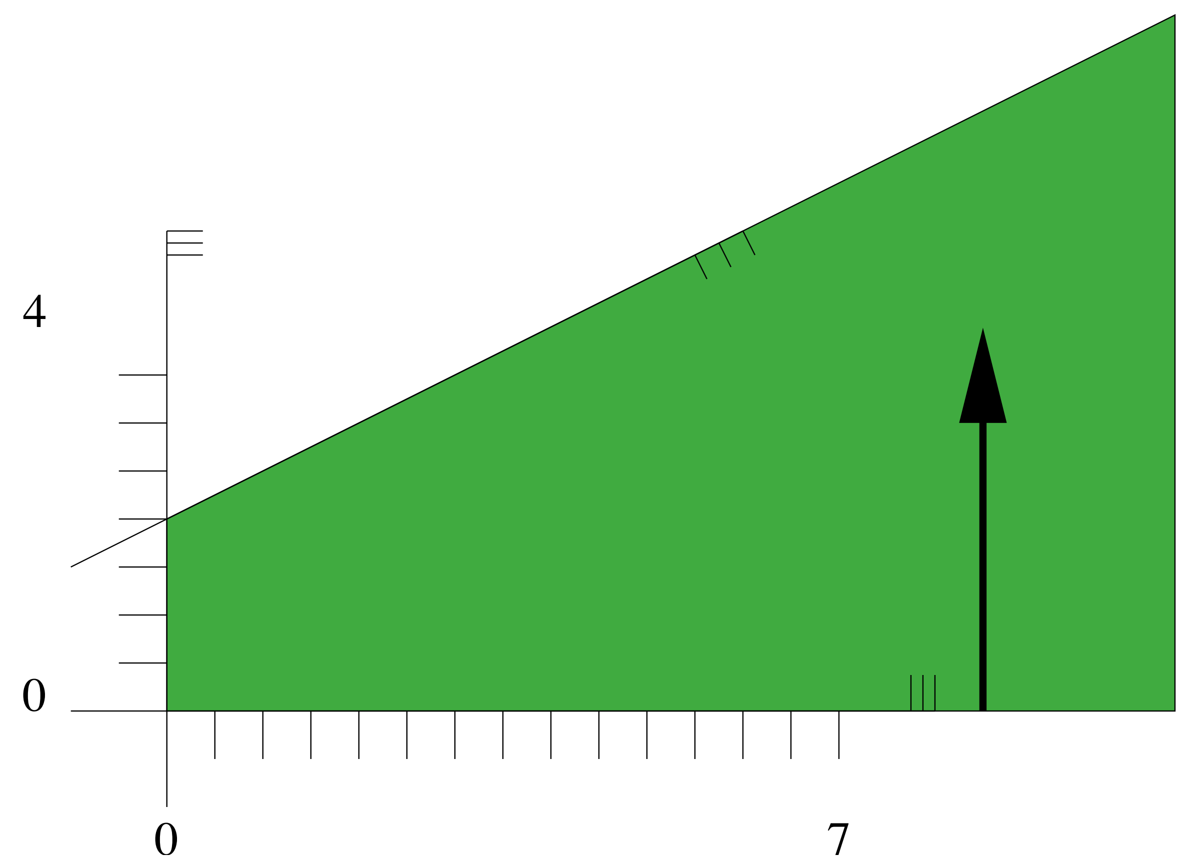
Linear Programming

Mathematical formulation

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

- **Unbounded linear programs:**

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & \cancel{x + y} & \cancel{\leq} & \cancel{7} \\
& -x + 2y & \leq & 4 \\
& x & \geq & 0 \\
& y & \geq & 0 \\
& \cancel{y} & \cancel{\leq} & \cancel{4}
\end{array}
$$



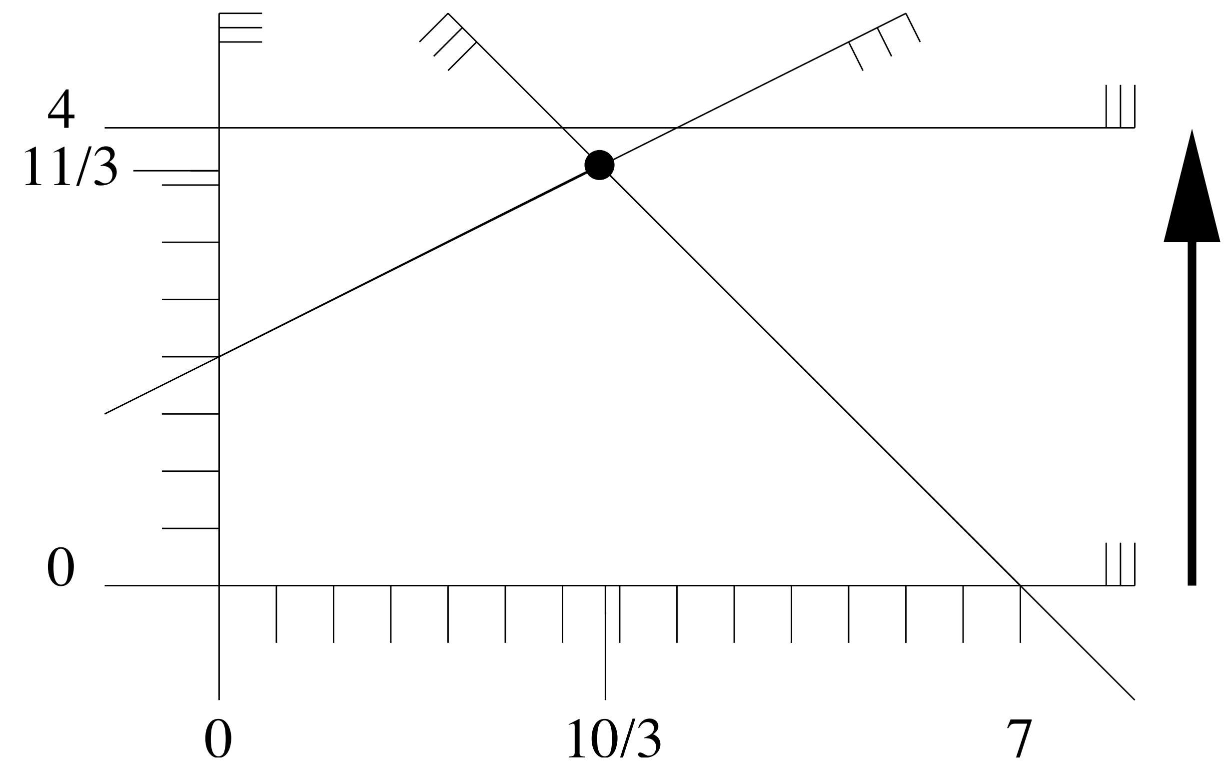
Linear Programming

Mathematical formulation

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

- **Infeasible linear programs:**

$$\begin{array}{llcl}
\text{minimize} & -32y+64 & & \\
\text{subject to} & x+y & \leq & 7 \\
 & -x+2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
y \geq 4 & \cancel{y} & \cancel{\leq} & \cancel{4}
\end{array}$$



Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
\quad \Rightarrow \quad
\underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}}_{A}
\begin{pmatrix} x \\ y \end{pmatrix}
\leq
\underbrace{\begin{pmatrix} 7 \\ 4 \\ 0 \\ 0 \\ 4 \end{pmatrix}}_{b}
$$

Linear Programming

Mathematical formulation

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

$$\begin{array}{lrcl}
\text{minimize} & -32y+64 & & \\
\text{subject to} & x+y & \leq & 7 \\
 & -x+2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
\quad \Rightarrow \quad
\underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}}_{A}
\begin{pmatrix} x \\ y \end{pmatrix}
\leq
\underbrace{\begin{pmatrix} 7 \\ 4 \\ 0 \\ 0 \\ 4 \end{pmatrix}}_{b}$$

Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
\quad \Rightarrow \quad
\underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}}_{A}
\begin{pmatrix} x \\ y \end{pmatrix}
\leq
\underbrace{\begin{pmatrix} 7 \\ 4 \\ 0 \\ 0 \\ 4 \end{pmatrix}}_{b}
$$

Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$
\begin{array}{lrcl}
\text{minimize} & -32y+64 & & \\
\text{subject to} & x+y & \leq & 7 \\
 & -x+2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
\quad \Rightarrow \quad
\underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}}_{A}
\begin{pmatrix} x \\ y \end{pmatrix}
\leq
\underbrace{\begin{pmatrix} 7 \\ 4 \\ 0 \\ 0 \\ 4 \end{pmatrix}}_{b}
$$

Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5)**:**

$$
\begin{array}{lrcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & -x \leq 0 \quad \cancel{x} & \cancel{\geq} & \cancel{0} \\
 & -y \leq 0 \quad \cancel{y} & \cancel{\geq} & \cancel{0} \\
 & y & \leq & 4
\end{array}
\quad \Rightarrow \quad
\underbrace{\begin{bmatrix} 1 & 1 \\ -1 & 2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}}_{A}
\begin{bmatrix} x \\ y \end{bmatrix}
\leq
\underbrace{\begin{bmatrix} 7 \\ 4 \\ 0 \\ 0 \\ 4 \end{bmatrix}}_{b}
$$

Linear Programming

Mathematical formulation

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

- **Example** (*n*=2, *m*=5):

$$\begin{array}{lrcl} \text{minimize} & \boxed{-32y+64} & & \\ \text{subject to} & x+y & \le & 7 \\ & -x+2y & \le & 4 \\ & -x \le 0 \quad \cancel{x} & \cancel{\ge} & \cancel{0} \\ & -y \le 0 \quad \cancel{y} & \cancel{\ge} & \cancel{0} \\ & y & \le & 4 \end{array}$$

$$\overset{c^T}{\begin{pmatrix}0\\-32\end{pmatrix}}\begin{pmatrix}x & y\end{pmatrix} + \overset{c_0}{\begin{pmatrix}64\end{pmatrix}}$$

$$\underset{A}{\begin{pmatrix}1 & 1\\-1 & 2\\-1 & 0\\0 & -1\\0 & 1\end{pmatrix}}\begin{pmatrix}x\\y\end{pmatrix} \le \underset{b}{\begin{pmatrix}7\\4\\0\\0\\4\end{pmatrix}}$$

Linear Programming

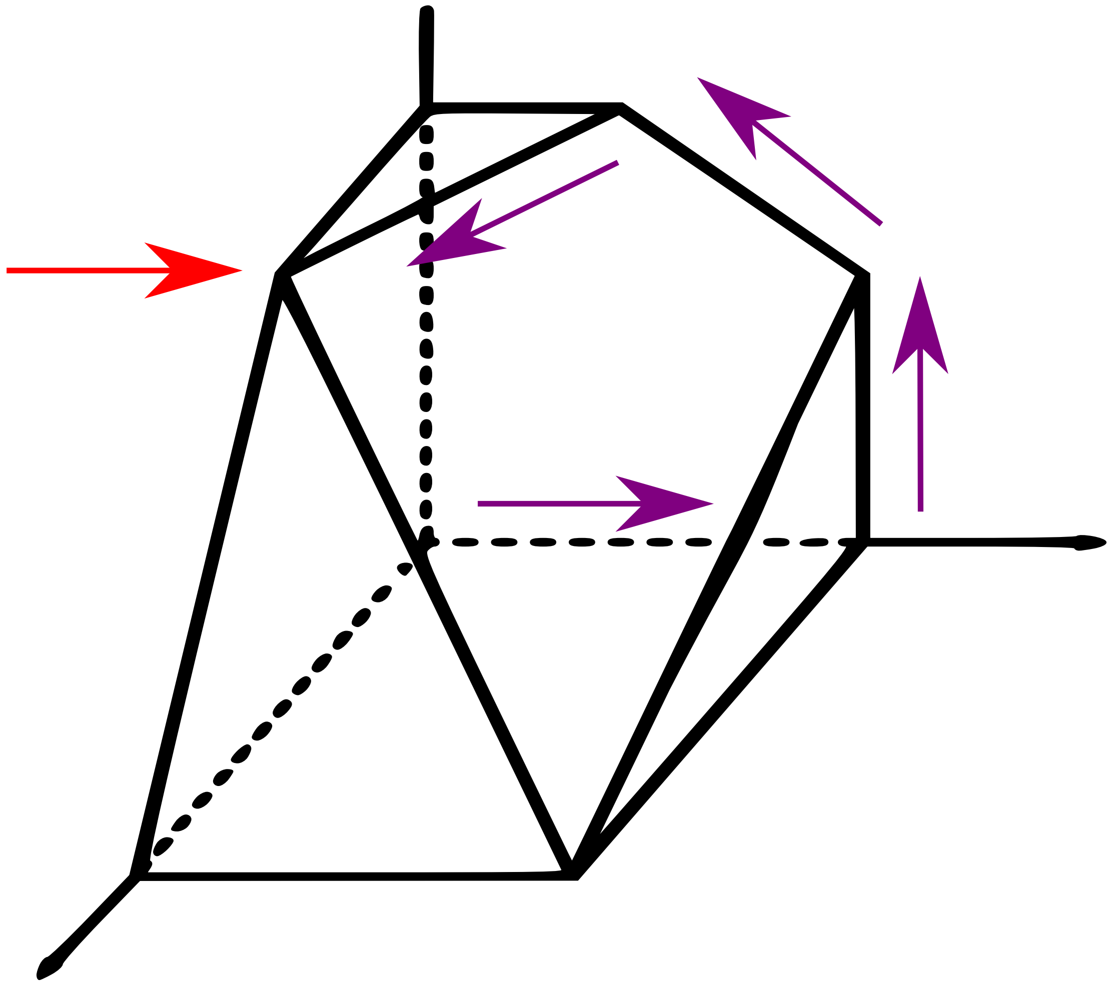
Mathematical formulation

- **General form of LP:**

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

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

Linear Programming



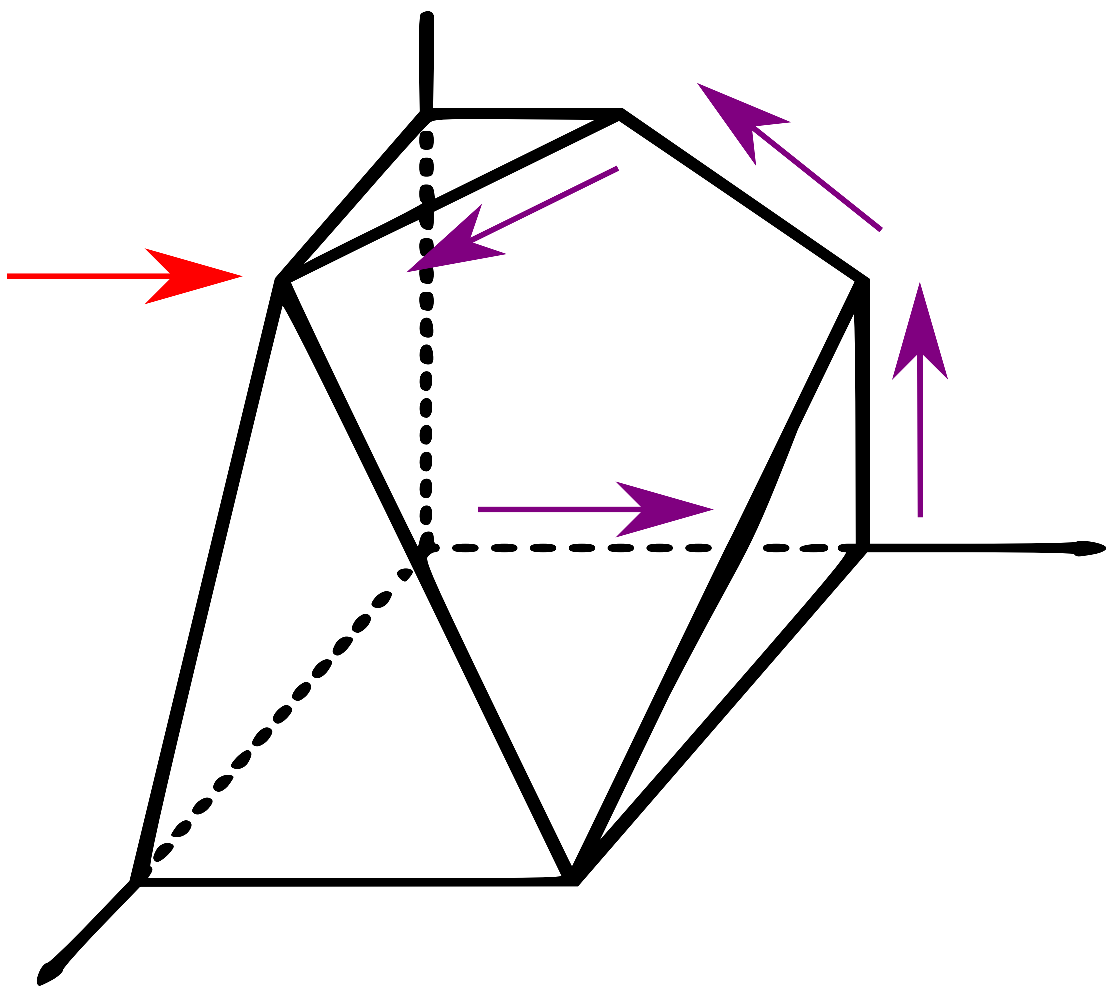
$$
\begin{array}{ll}
\text{minimize} & c^T x + c_0 \\
\text{subject to} & Ax \substack{\geq \\ =\\ \leq} b \\
 & l \leq x \leq u
\end{array}
$$

- The constraints define a convex polyhedron

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

Linear Programming

$$\begin{array}{ll} \text{minimize} & c^T x + c_0 \\ \text{subject to} & Ax \gtreqless b \\ & l \le x \le u \end{array}$$



- The constraints define a convex polyhedron
$$P = \{x | Ax \le b\}$$
- Simplex-type algorithms walk along the edges of P to vertices with better objective value.
- To decide on the next edge we use a *pivot rule.*

Linear Programming

Complexity of Simplex-type algorithms with *n* variables and *m* constraints.

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

- The constraints define a convex polyhedron
$$P = \{x | Ax \leq b\}$$
- Simplex-type algorithms walk along the edges of P to vertices with better objective value.
- To decide on the next edge we use a *pivot rule.*

Linear Programming

Complexity of Simplex-type algorithms with *n* variables and *m* constraints.

- Exponential running time in the worst case! (exp. on $min\{m,n\}$)

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

- The constraints define a convex polyhedron
$$P = \{x | Ax \leq b\}$$
- Simplex-type algorithms walk along the edges of P to vertices with better objective value.
- To decide on the next edge we use a *pivot rule*.

Linear Programming

Complexity of Simplex-type algorithms with *n* variables and *m* constraints.

- Exponential running time in the worst case! (exp. on *min*{*m*,*n*})

- O(*max*{*m*, *n*}) if *min*{*m*,*n*} is *O*(1) (hidden constant exponential in *min*{*m*,*n*})

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

- The constraints define a convex polyhedron

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

- Simplex-type algorithms walk along the edges of *P* to vertices with better objective value.
- To decide on the next edge we use a *pivot rule.*

Linear Programming

Complexity of Simplex-type algorithms with *n* variables and *m* constraints.

- Exponential running time in the worst case! (exp. on *min{m,n}*)

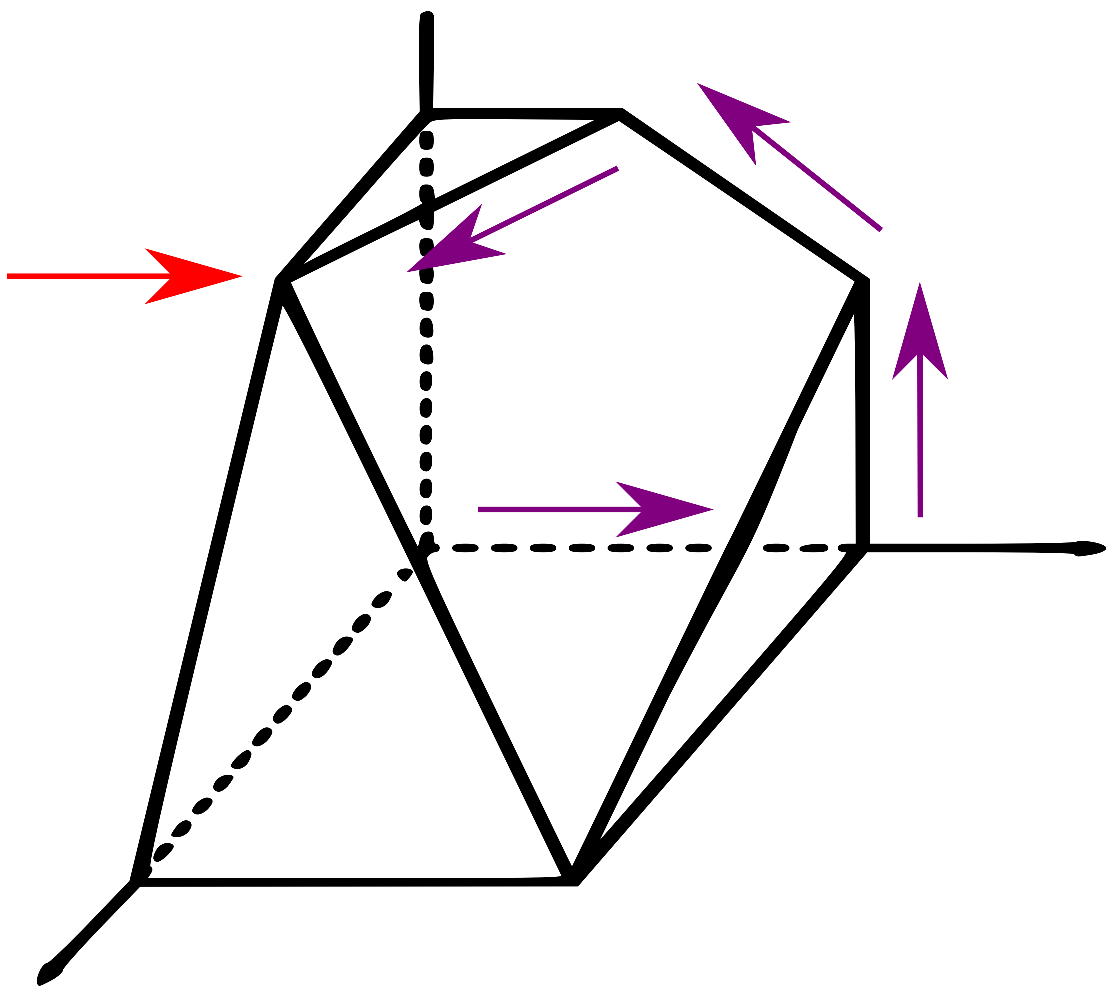
- O(*max{m, n}*) if *min{m,n}* is *O*(1) (hidden constant exponential in *min{m,n}*)

- CGAL: Gives exact solutions! Works well as long as *min{m,n}* ≤ 50 (or up to 100)

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

- The constraints define a convex polyhedron
$$P = \{x | Ax \leq b\}$$
- Simplex-type algorithms walk along the edges of P to vertices with better objective value.
- To decide on the next edge we use a *pivot rule.*

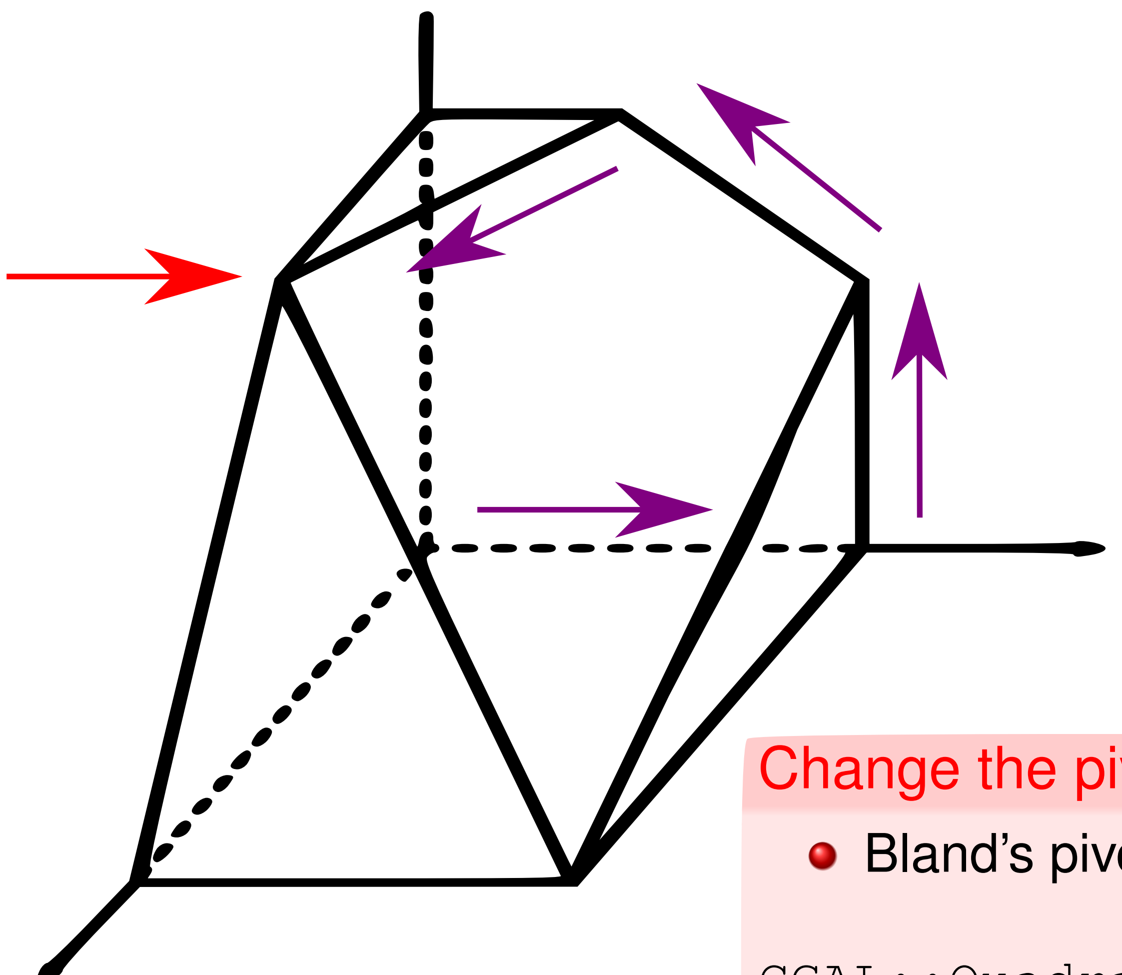
Linear Programming



The default pivot rule strategy might fail:

- it is deterministic, so it picks the same improving edge every time.
- If the algorithm reaches a vertex for the second time, then cycles forever.

Linear Programming



The default pivot rule strategy might fail:

- it is deterministic, so it picks the same improving edge every time.
- If the algorithm reaches a vertex for the second time, then cycles forever.

Change the pivot rule

- Bland's pivot rule avoids cycling (but it is slower. . .)

```
CGAL::Quadratic_program_options options;
options.set_pricing_strategy(CGAL::QP_BLAND);
Solution s = CGAL::SOLVER(program, ET(), options);
```

Linear Programming

- **General form of LP in CGAL:**

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

≤, =, or ≥ (individually for each constraint)

variables

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

objective function

shift

lower and upper bounds

right-hand side

constraint matrix

Linear Programming ... in CGAL

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

- **Preamble:** Choice of input type and exact internal number type

Gnu Multi-precision Library (GMP)

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

// choose exact integral type
typedef CGAL::Gmpz ET;




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

input type

exact internal type

for linear *and* quadratic programs

GMP used internally

Linear Programming ... in CGAL

$$\text{minimize} \quad -32y + 64 = c^T x + c_0$$
$$\text{subject to} \quad Ax \le b \begin{cases} x + y & \le & 7 \\ -x + 2y & \le & 4 \\ x & \ge & 0 \\ y & \ge & 0 \\ y & \le & 4 \end{cases}$$

- **Preamble:** Choice of input type and exact internal number type

Gnu Multi-precision Library (GMP)

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

// choose exact integral type
typedef CGAL::Gmpz ET;

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

input type

Inside, the LP solver uses quotients of the data type specified here.

exact internal type

for linear *and* quadratic programs

GMP used internally

Linear Programming ... in CGAL

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

- **Preamble:** Choice of input type and exact internal number type

Gnu Multi-precision Library (GMP)

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

// choose exact integral type
typedef CGAL::Gmpz ET;




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

input type

exact internal type

for linear *and* quadratic programs

GMP used internally

Linear Programming ... in CGAL

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

- **Setup:** Enter the program data

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);

  // now set the non-default entries
  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
  lp.set_u(Y, true, 4);                                   //      y <= 4
  lp.set_c(Y, -32);                                       // -32y
  lp.set_c0(64);                                          // +64
```

$l =$ (0,0,..., 0)

$u = (\infty,\infty,\ldots, \infty)$

last argument: value

variable index (0,1,...)

constraint index (0,1,...)

Linear Programming ... in CGAL

$$
\begin{array}{ll}
\text{minimize} & -32y + 64 = c^T x + c_0 \\
\text{subject to} & \\
Ax \le b & \left\{ \begin{array}{rcl} \boxed{x + y \le 7} \\ -x + 2y & \le & 4 \\ x & \ge & 0 \\ y & \ge & 0 \\ y & \le & 4 \end{array} \right.
\end{array}
$$

- **Setup:** Enter the program data

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);

  // now set the non-default entries
  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
  lp.set_u(Y, true, 4);                                   //       y <= 4
  lp.set_c(Y, -32);                                       // -32y
  lp.set_c0(64);                                          // +64
```

$l =$ $(0,0,\ldots,0)$ $u = (\infty,\infty,\ldots,\infty)$

variable index (0,1,...)

constraint index (0,1,...)

last argument: value

Linear Programming ... in CGAL

$$
\begin{array}{ll}
\text{minimize} & -32y + 64 = c^T x + c_0 \\
\text{subject to} & \\
\end{array}
$$

$$
Ax \le b \left\{ \begin{array}{rcl}
x + y & \le & 7 \\
\boxed{-x + 2y} & \boxed{\le} & \boxed{4} \\
x & \ge & 0 \\
y & \ge & 0 \\
y & \le & 4
\end{array} \right.
$$

- **Setup:** Enter the program data

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);

  // now set the non-default entries
  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
  lp.set_u(Y, true, 4);                                   //       y <= 4
  lp.set_c(Y, -32);                                       // -32y
  lp.set_c0(64);                                          // +64
```

$l =$ $(0,0,\ldots, 0)$ $u = (\infty,\infty,\ldots, \infty)$

variable index (0,1,...)

constraint index (0,1,...)

last argument: value

Linear Programming ... in CGAL

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

- **Setup:** Enter the program data

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);

  // now set the non-default entries
  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
  lp.set_u(Y, true, 4);                                   //       y <= 4
  lp.set_c(Y, -32);                                       // -32y
  lp.set_c0(64);                                          // +64
```

$l = (0,0,\ldots,0)$

$u = (\infty,\infty,\ldots,\infty)$

variable index (0,1,...)

constraint index (0,1,...)

last argument: value

Linear Programming
... in CGAL

minimize $\boxed{-32y + 64 = c^T x + c_0}$

subject to

$$Ax \le b \left\{ \begin{array}{rcl} x + y & \le & 7 \\ -x + 2y & \le & 4 \\ x & \ge & 0 \\ y & \ge & 0 \\ y & \le & 4 \end{array} \right.$$

- **Setup:** Enter the program data

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);

  // now set the non-default entries
  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
  lp.set_u(Y, true, 4);                                    //       y <= 4
  lp.set_c(Y, -32);                                        // -32y
  lp.set_c0(64);                                           // +64
```

$l =$ $(0,0,\ldots, 0)$

$u = (\infty,\infty,\ldots, \infty)$

variable index (0,1,...)

constraint index (0,1,...)

last argument: value

Linear Programming ... in CGAL

- **Solve:** Call the linear programming solver and output solution

```
 // 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;
  return 0;
}
```

independent verification

Linear Programming
… in CGAL

- **Solve:** Call the linear programming solver and output solution

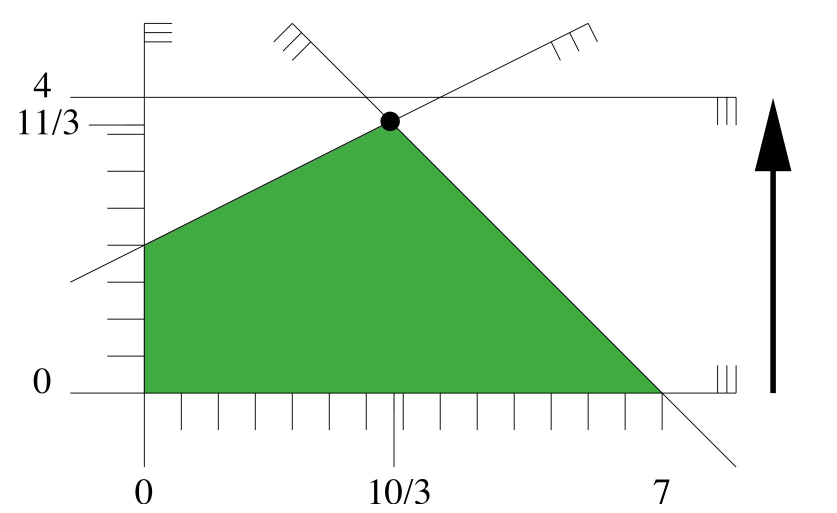
```
  // solve the program, using ET as the exact type
  Solution s = CGAL::solve_linear_program(lp, ET());
  assert (s.solves_linear_program(lp));

  // output solution
  std::cout << s;
  return 0;
}
```

independent verification

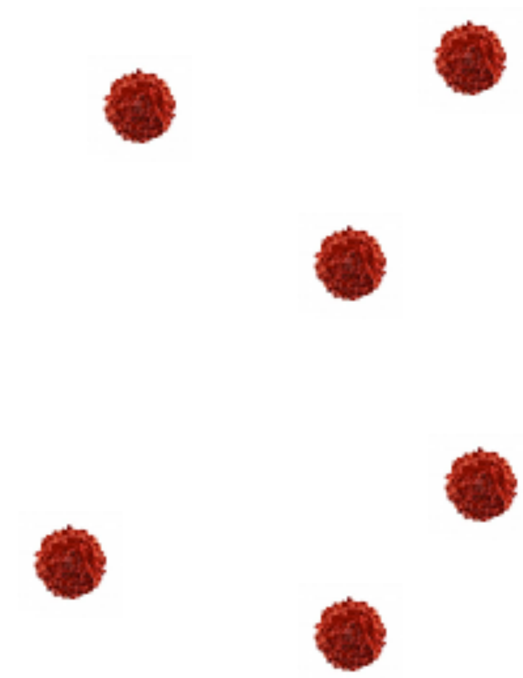
- **Output:**

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



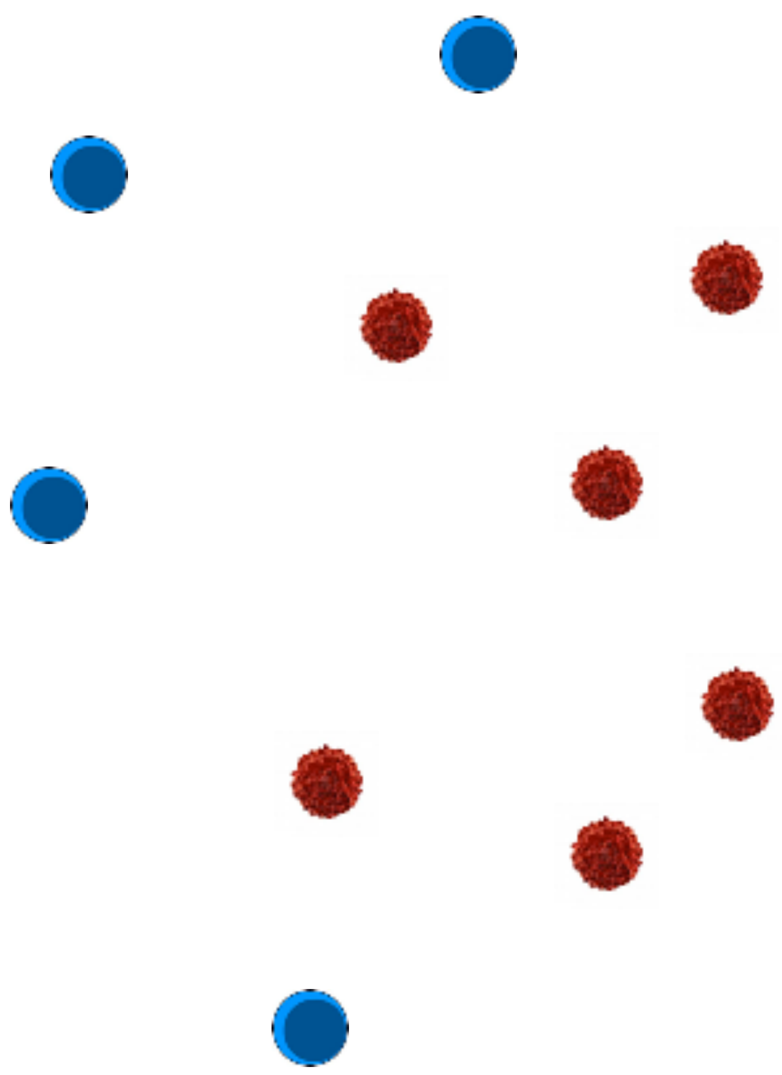
Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red)



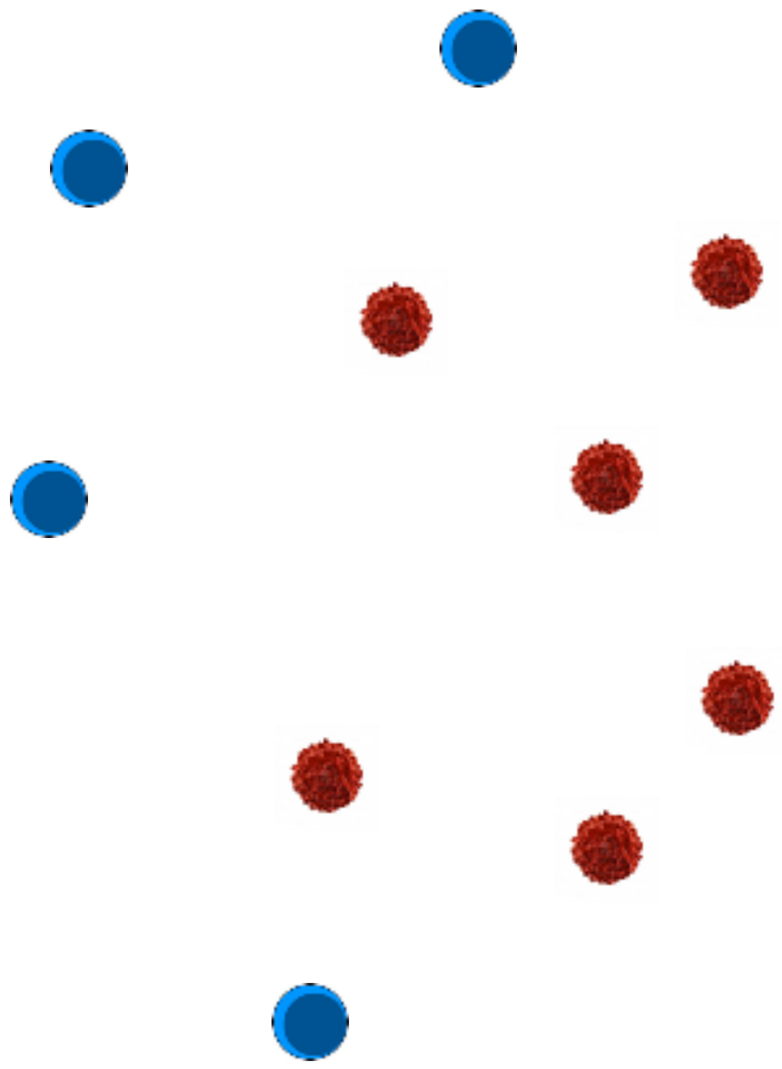
Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)



Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)
- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.

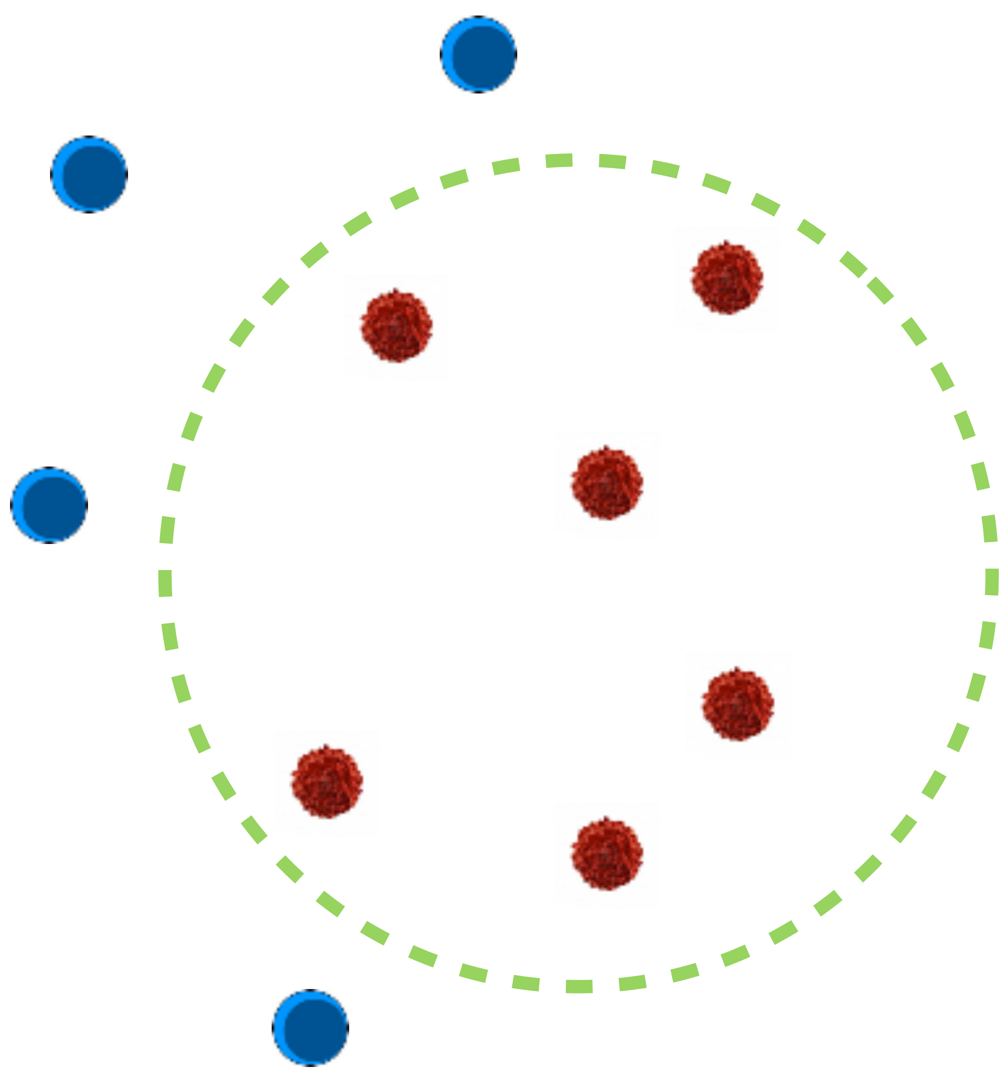


Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)

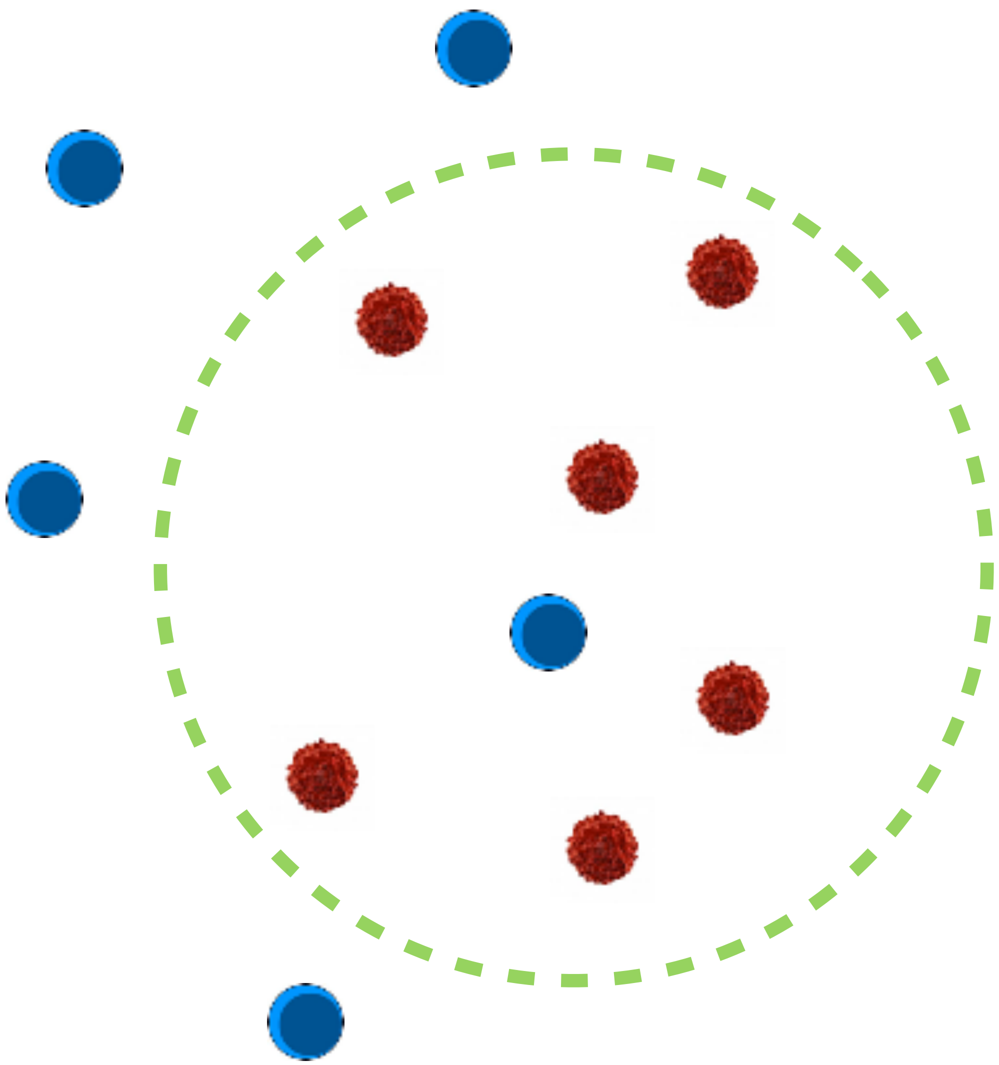
- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.

- This may be possible…



Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)
- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.
- This may be possible... or not.

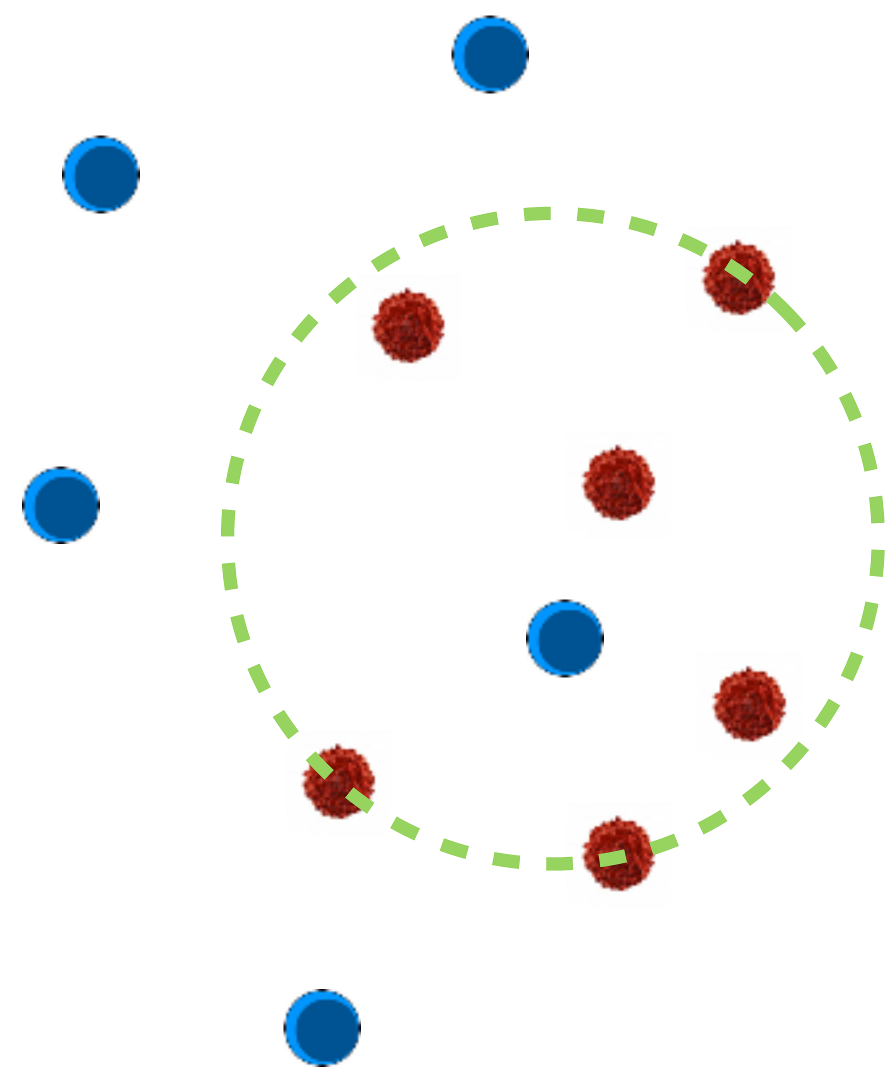


Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)

- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.

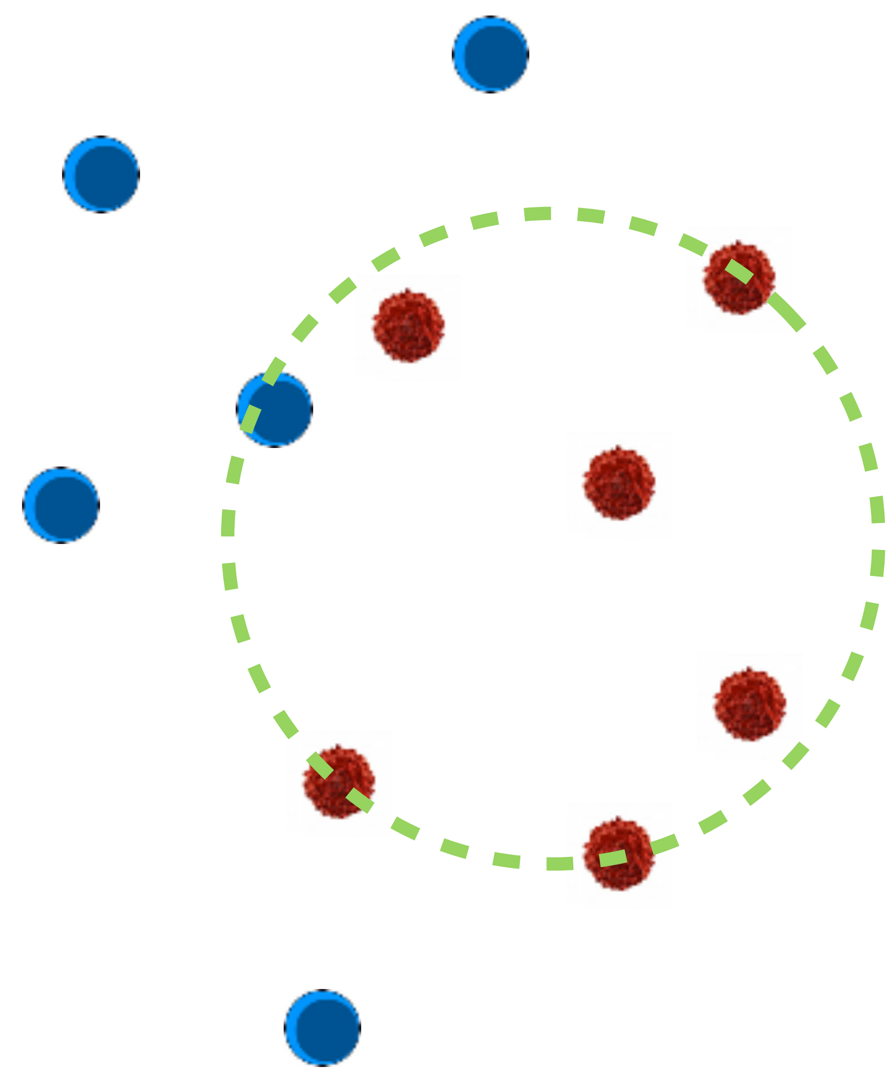
- This may be possible… or not.



- Minimum Enclosing circle is not enough to solve the problem.

Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)
- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.
- This may be possible… or not.



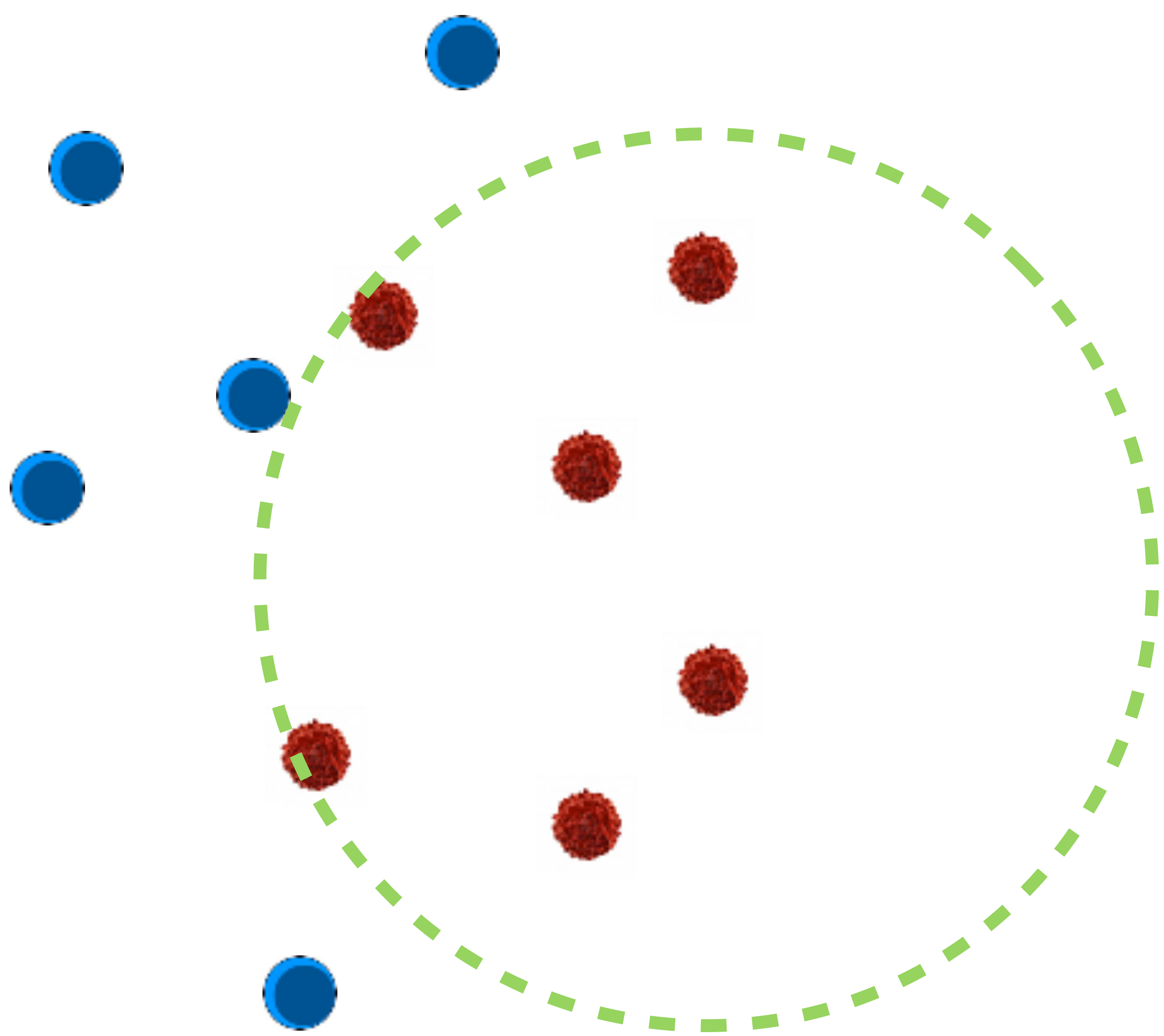
- Minimum Enclosing circle is not enough to solve the problem.

Linear Programming Application I: Cancer Therapy

- **Given:** locations of cancer cells (red) and healthy cells (blue)

- **Wanted:** center and radius of exposure so that all cancer cells are killed and all healthy cells are unaffected.

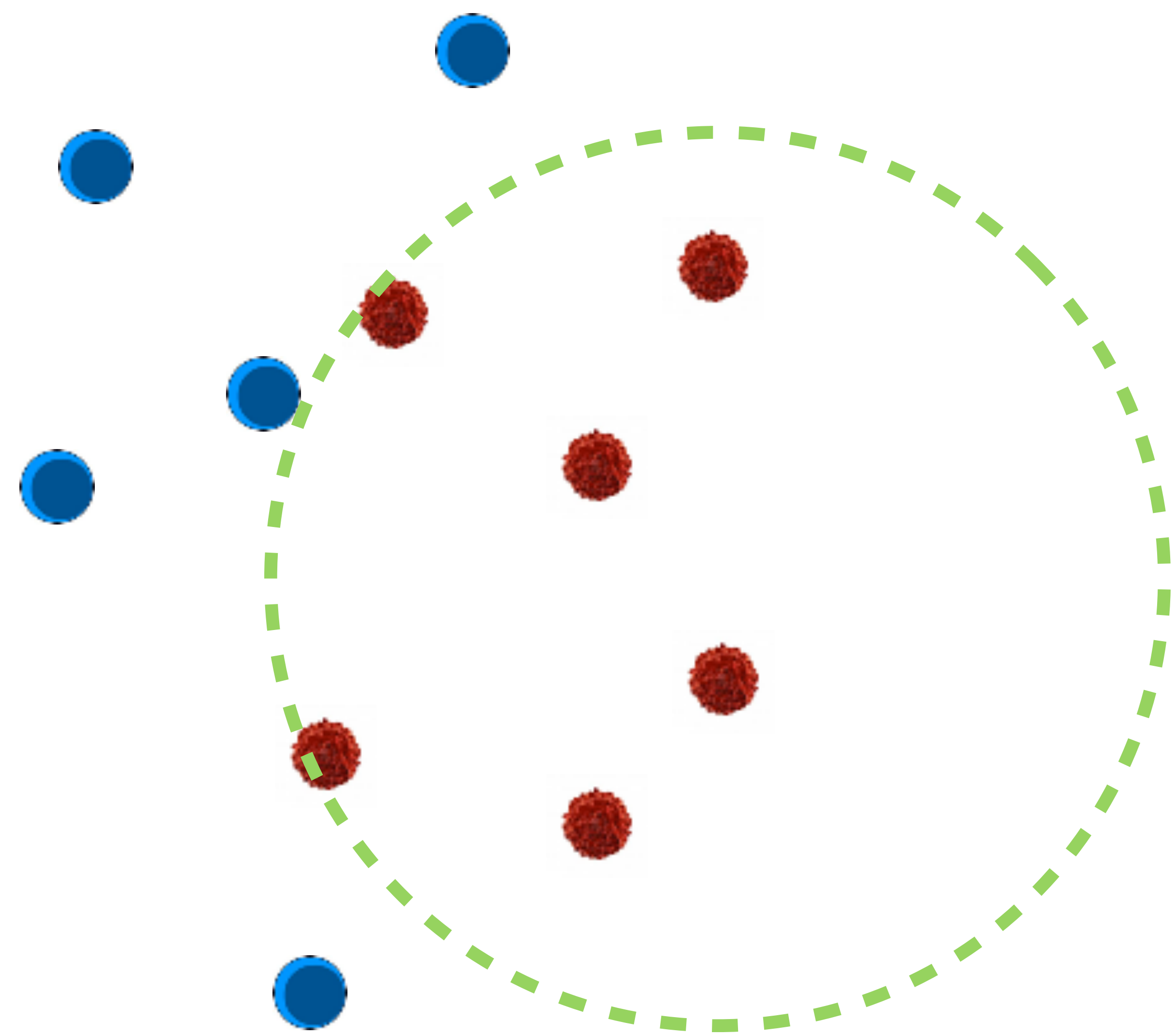
- This may be possible… or not.



- Minimum Enclosing circle is not enough to solve the problem.

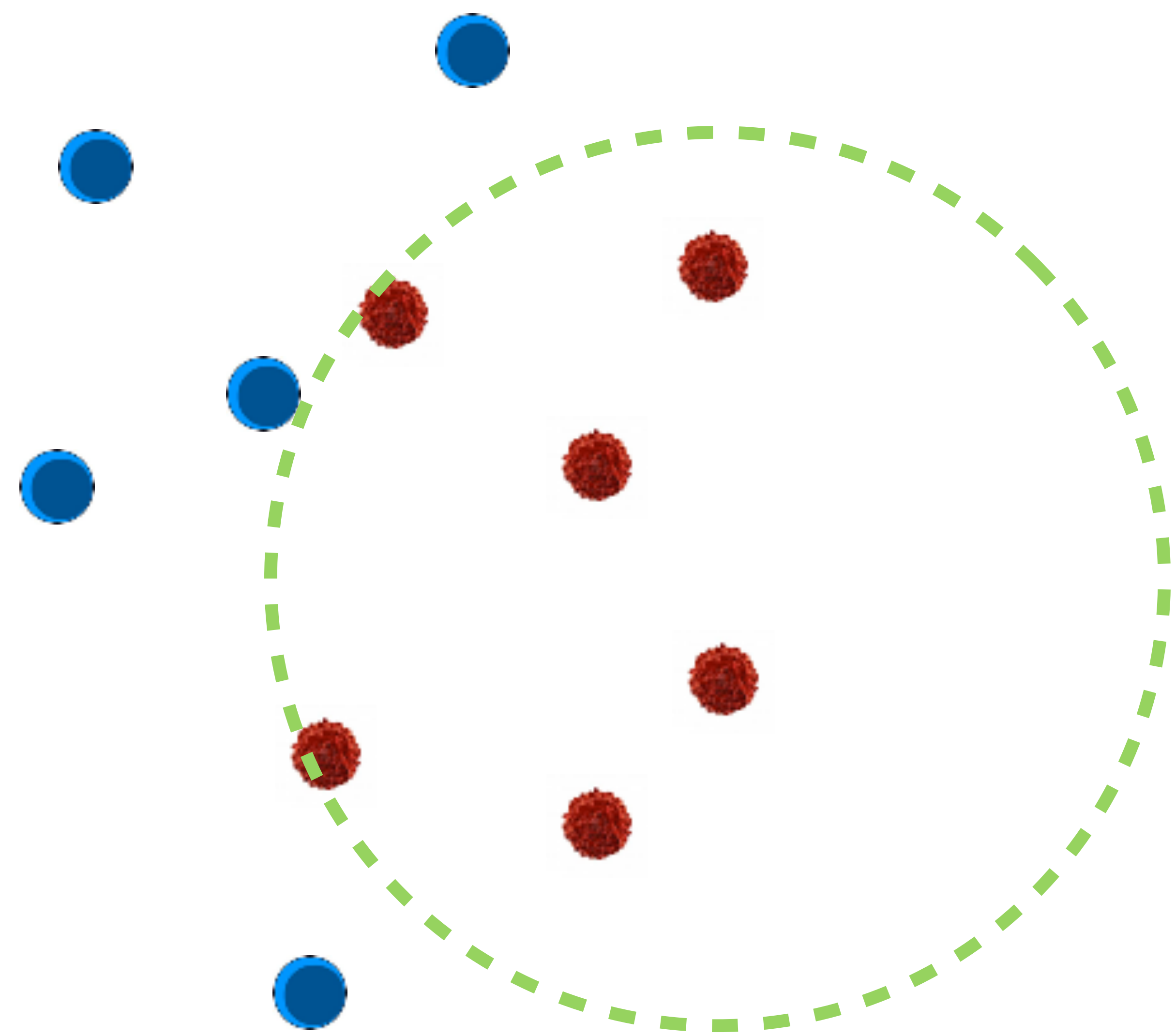
Linear Programming Application I: Cancer Therapy

- **The geometric problem:** Given two finite sets R and B in the plane, does there exist a disk that contains R and is disjoint from B?



Linear Programming Application I: Cancer Therapy

- **The geometric problem:** Given two finite sets *R* and *B* in the plane, does there exist a disk that contains *R* and is disjoint from *B*?
- **Variables:** Parameters used to define a circle.

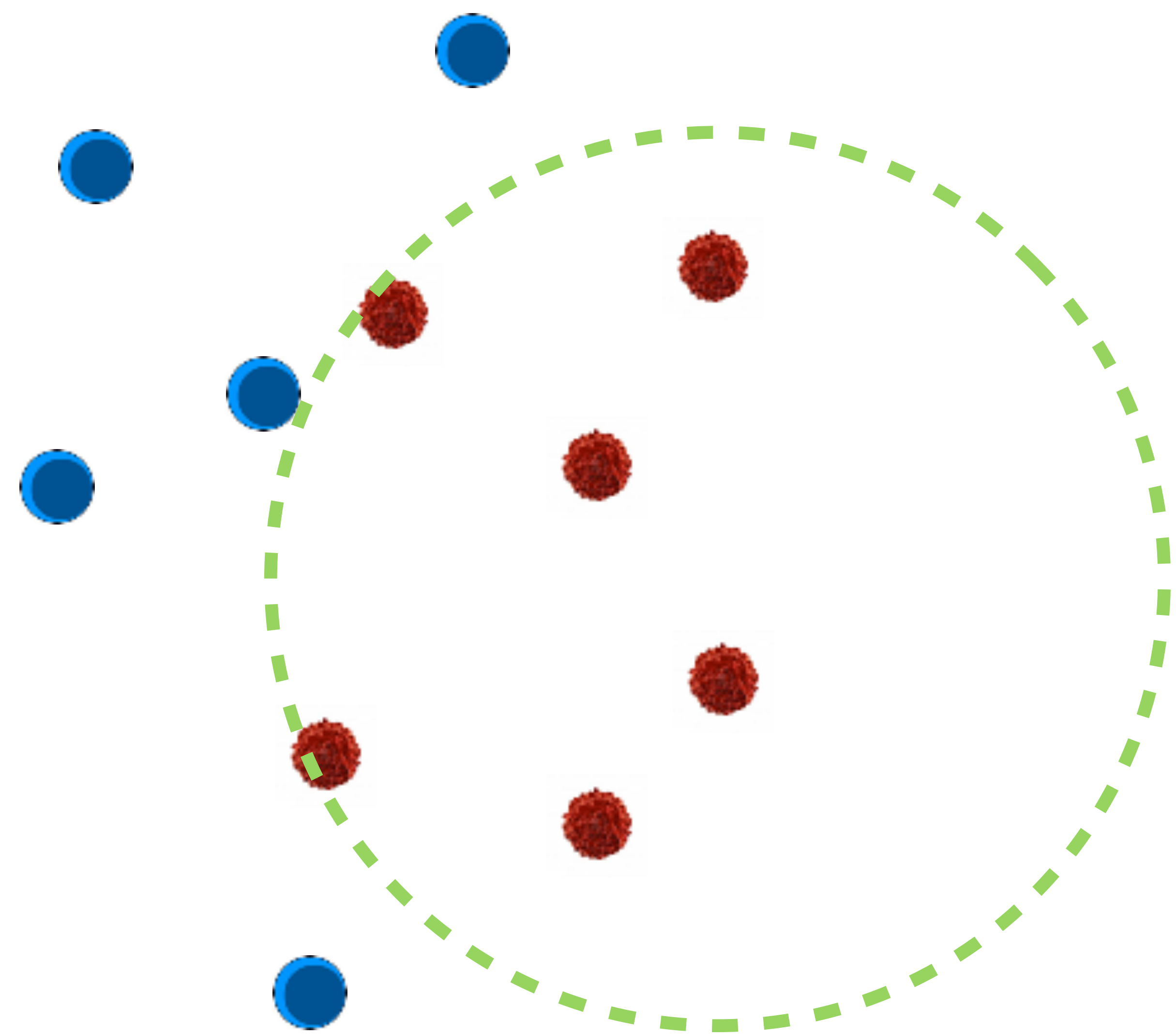


Linear Programming Application I: Cancer Therapy

- **The geometric problem:** Given two finite sets R and B in the plane, does there exist a disk that contains R and is disjoint from B?

- **Variables:** Parameters used to define a circle.

- **Constraints:** Blue points should be outside, red points inside.



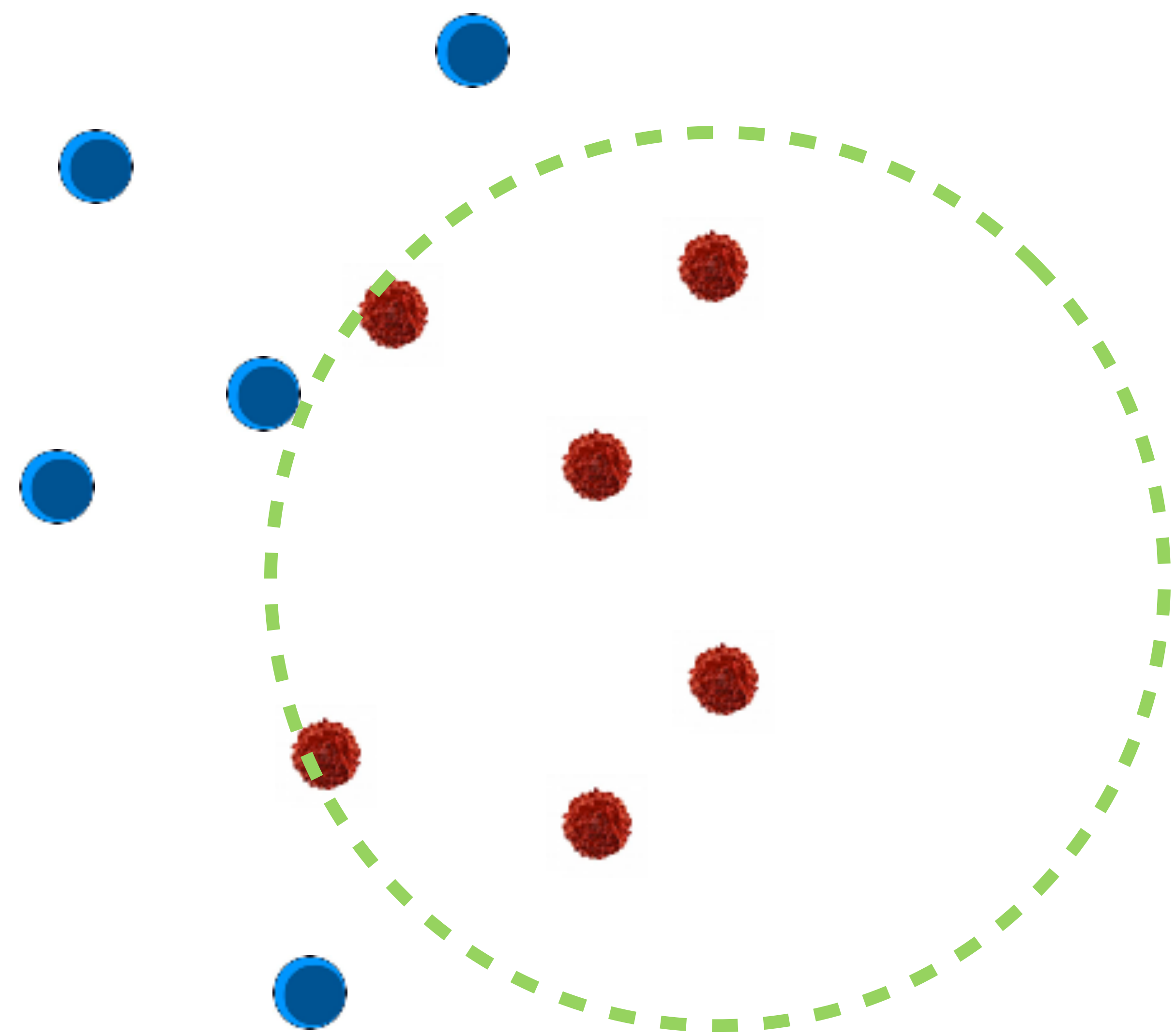
Linear Programming Application I: Cancer Therapy

- **The geometric problem:** Given two finite sets R and B in the plane, does there exist a disk that contains R and is disjoint from B?

- **Variables:** Parameters used to define a circle.

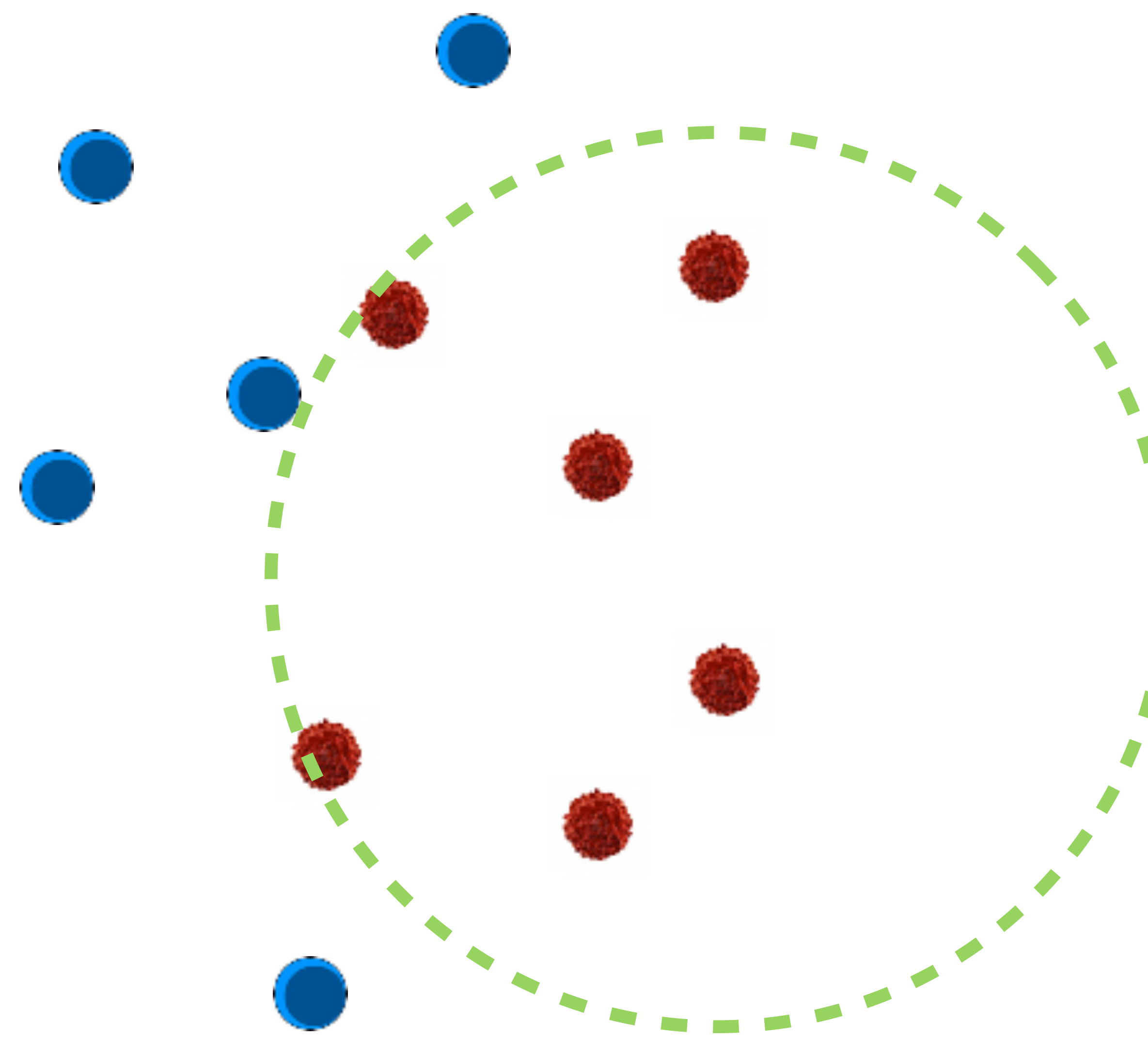
- **Constraints:** Blue points should be outside, red points inside.

- **Problem:** Constraints are not linear…

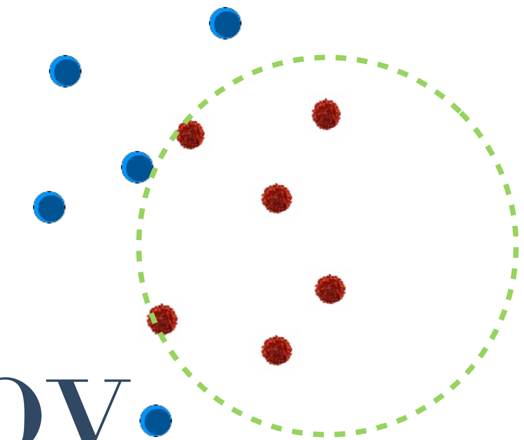


Linear Programming Application I: Cancer Therapy

- **The geometric problem:** Given two finite sets R and B in the plane, does there exist a disk that contains R and is disjoint from B?
- **Variables:** Parameters used to define a circle.
- **Constraints:** Blue points should be outside, red points inside.
- **Problem:** Constraints are not linear… or are they?

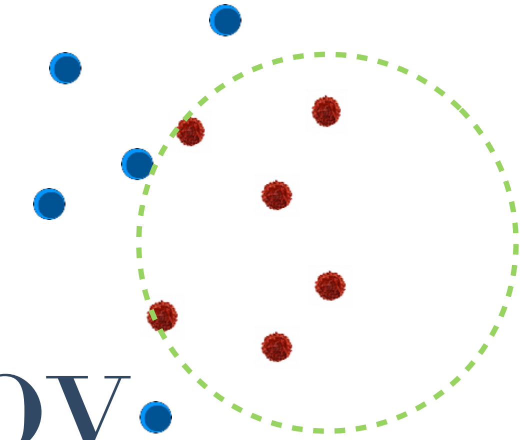


Linear Programming Application I: Cancer Therapy



- **Problem:** We want to represent the property of being inside of a circle as a linear constraint.

Linear Programming Application I: Cancer Therapy



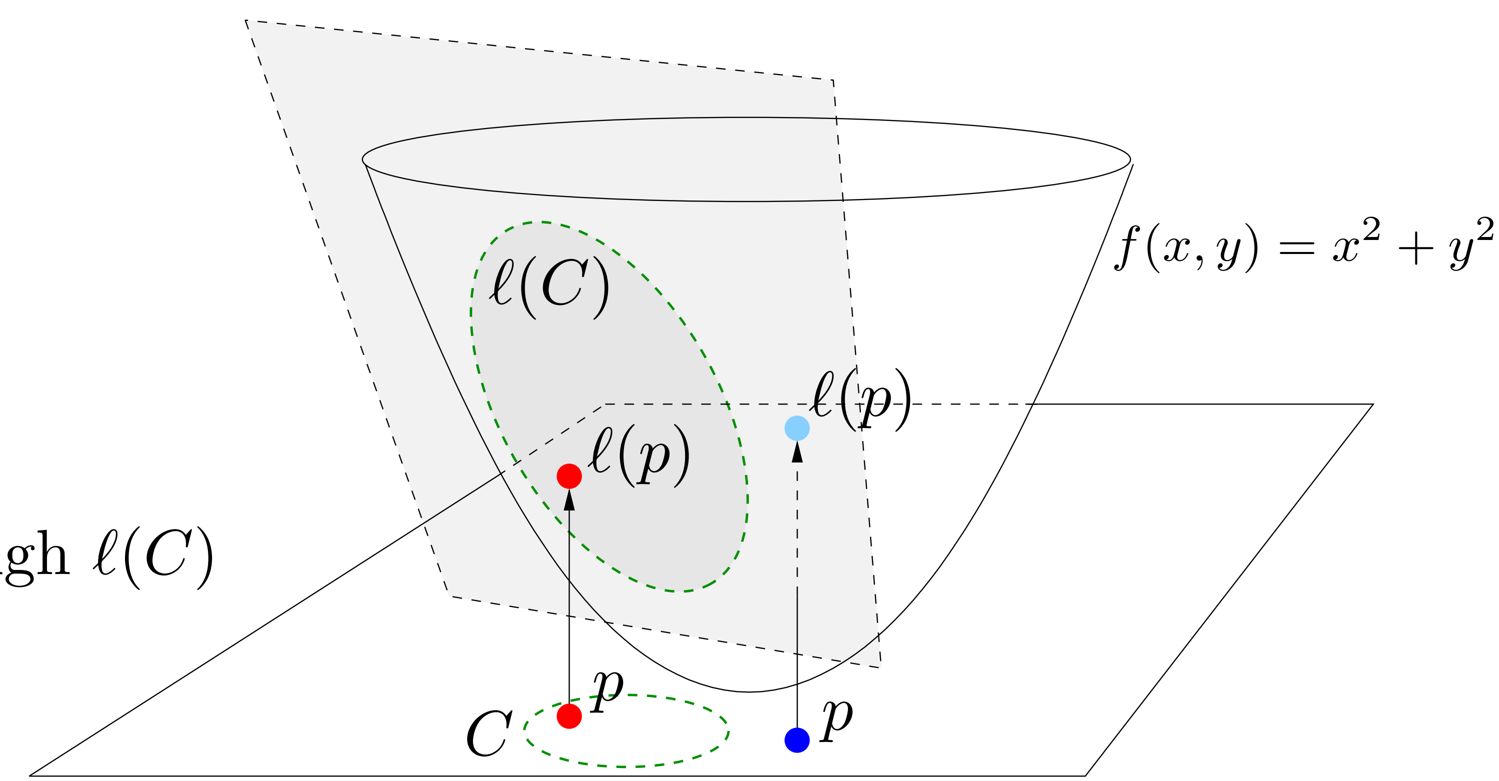
- **Problem:** We want to represent the property of being inside of a circle as a linear constraint.

- Apply *lifting map* $\ell : (x, y) \mapsto (x, y, x^2 + y^2)$

$$p \left\{ \begin{array}{l} \text{inside} \\ \text{on} \\ \text{outside} \end{array} \right\} C$$

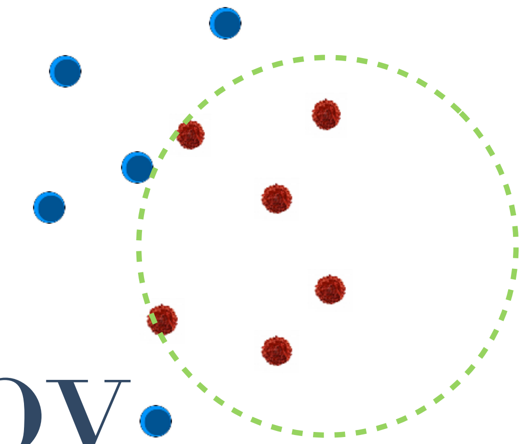
$$\Updownarrow$$

$$\ell(p) \left\{ \begin{array}{l} \text{below} \\ \text{on} \\ \text{above} \end{array} \right\} \text{the plane through } \ell(C)$$



[Ref: Section 2.3]

Linear Programming
Application I: Cancer Therapy



- **Problem:** We want to represent the property of being inside of a circle as a linear constraint.

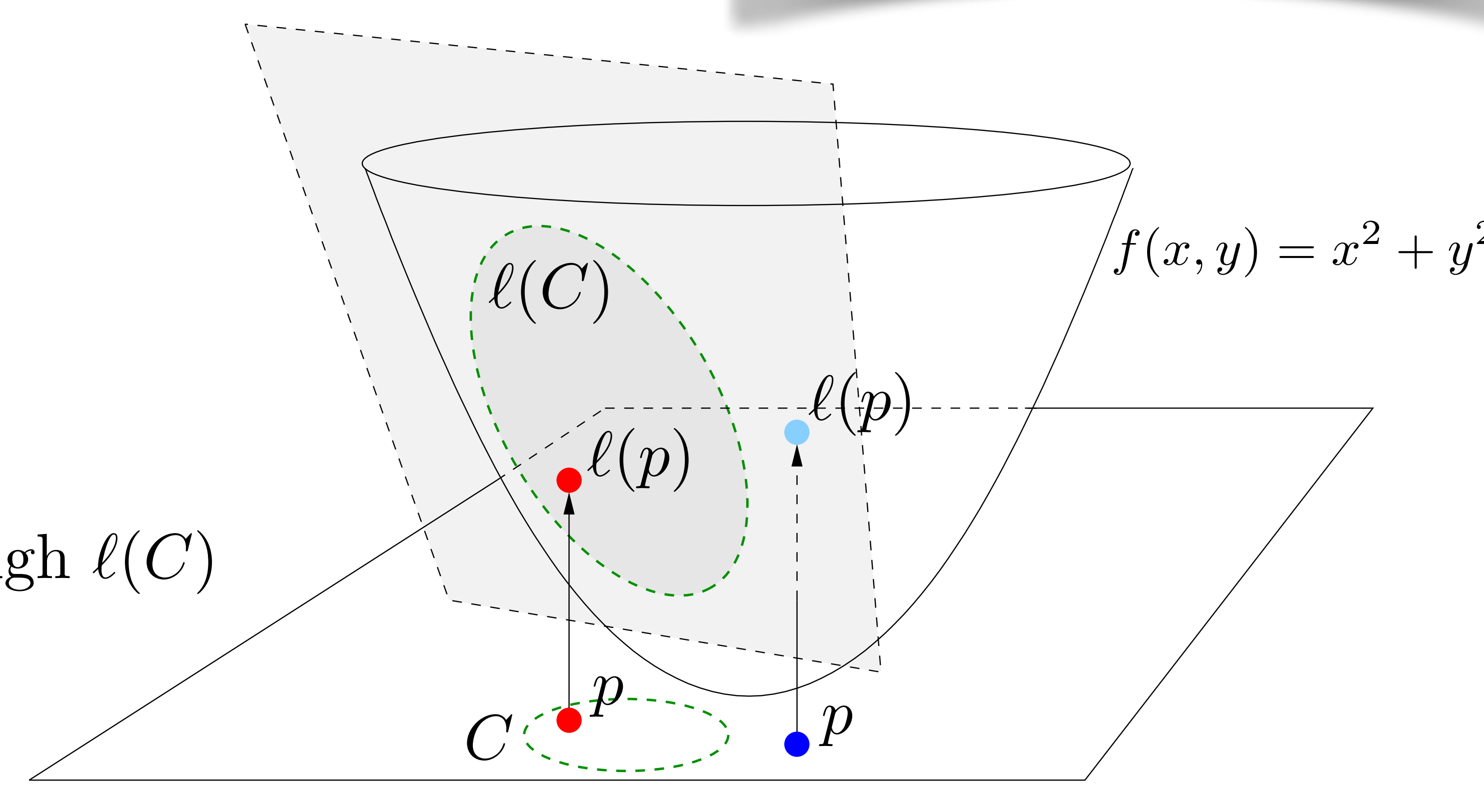
- Apply *lifting map* $\ell : (x, y) \mapsto (x, y, x^2 + y^2)$

Every circle has a corresponding plane

$$p \begin{Bmatrix} \text{inside} \\ \text{on} \\ \text{outside} \end{Bmatrix} C$$

$$\Updownarrow$$

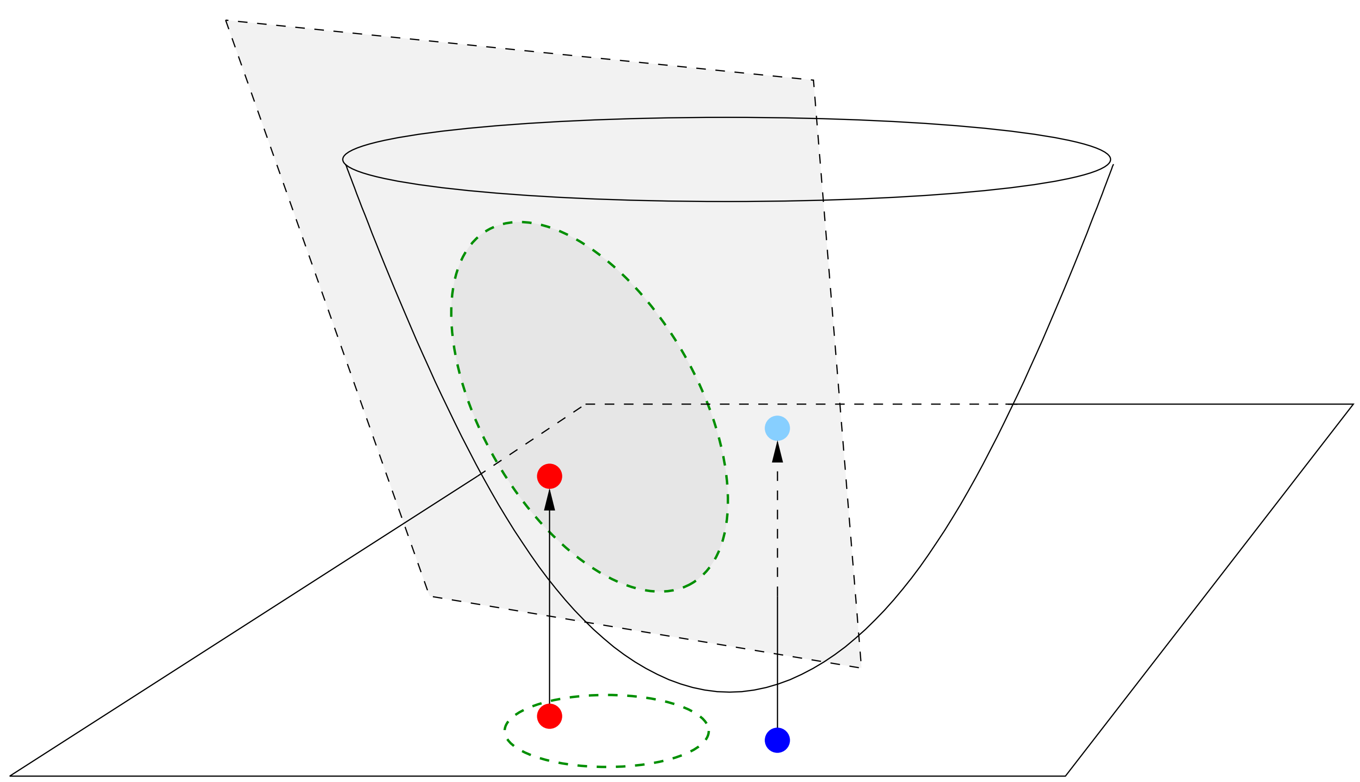
$$\ell(p) \begin{Bmatrix} \text{below} \\ \text{on} \\ \text{above} \end{Bmatrix} \text{the plane through } \ell(C)$$



[Ref: Section 2.3]

Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?



Linear Programming Application I: Cancer Therapy

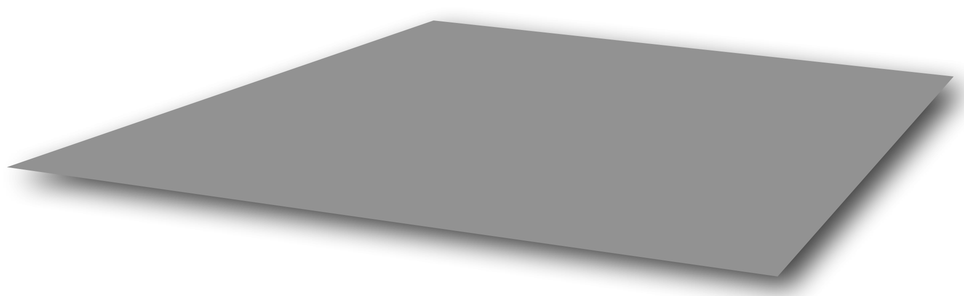
- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?
- This can be solved with linear programming!

Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

plane: $z = \alpha x + \beta y + \gamma$



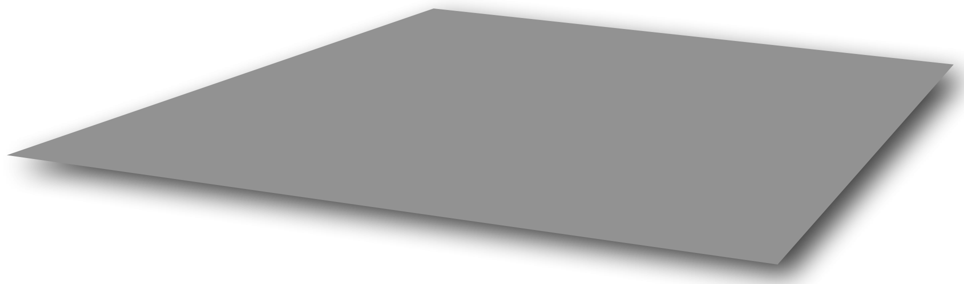
Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, δ ($\delta > 0$) such that...

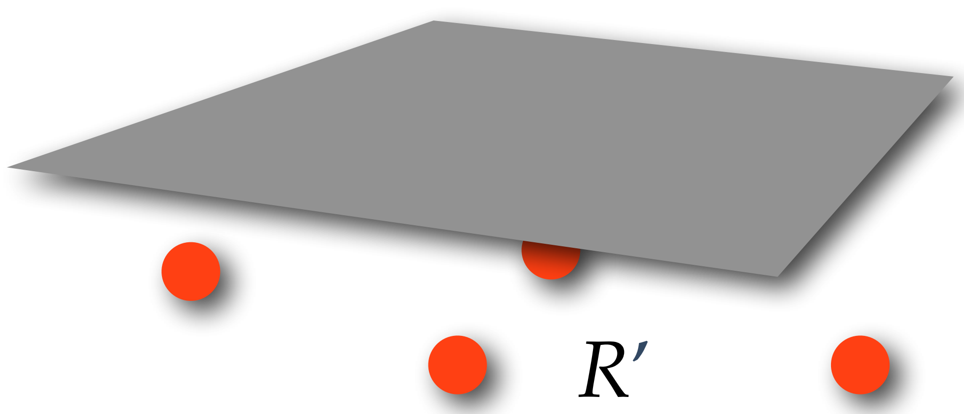
plane: $z = \alpha x + \beta y + \gamma$



Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?
- This can be solved with linear programming!
- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

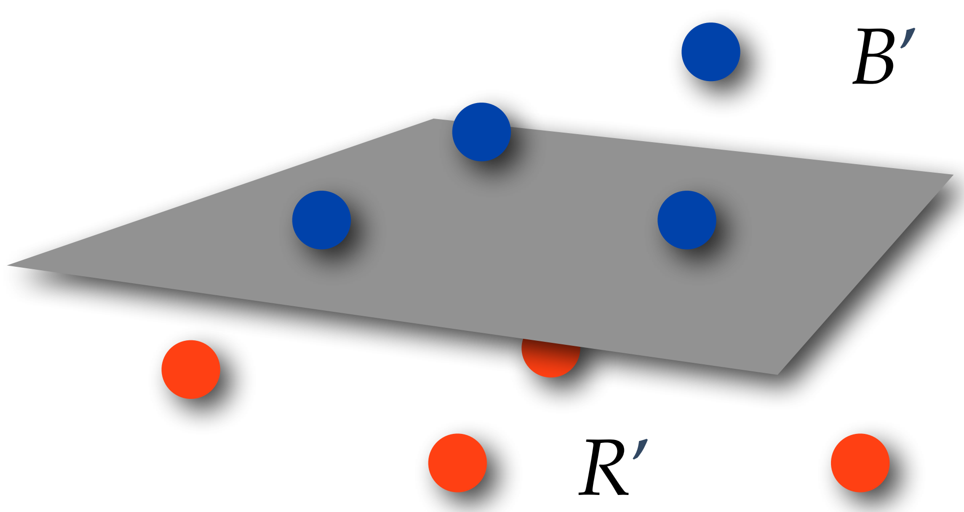
Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

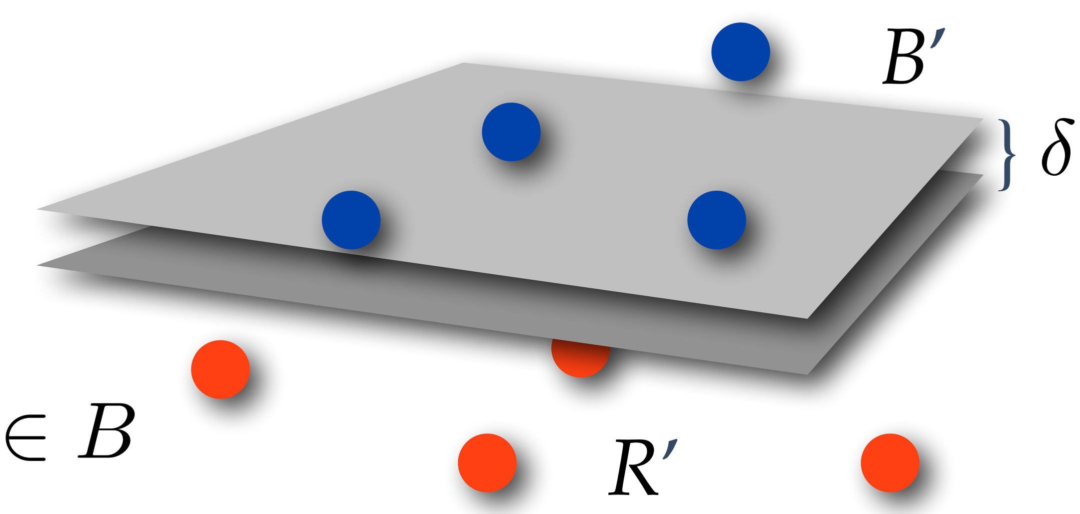
Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B$$

$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

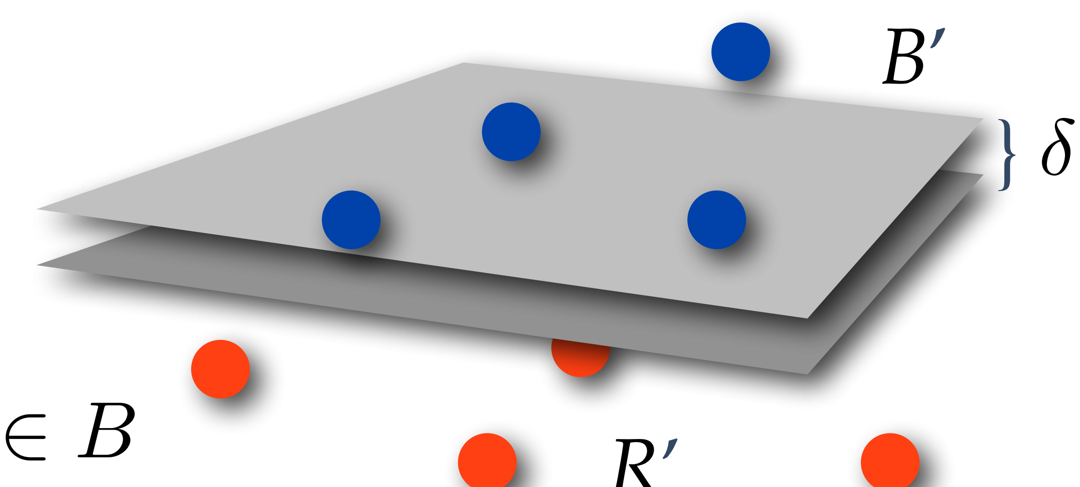
Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, δ > 0 such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B$$

$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

These are linear constraints!

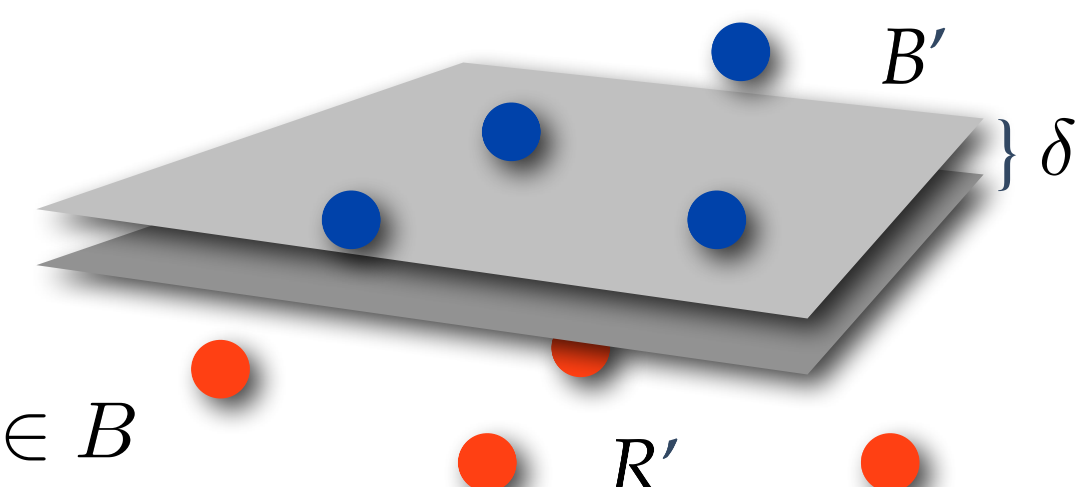
Linear Programming
Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B$$

$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

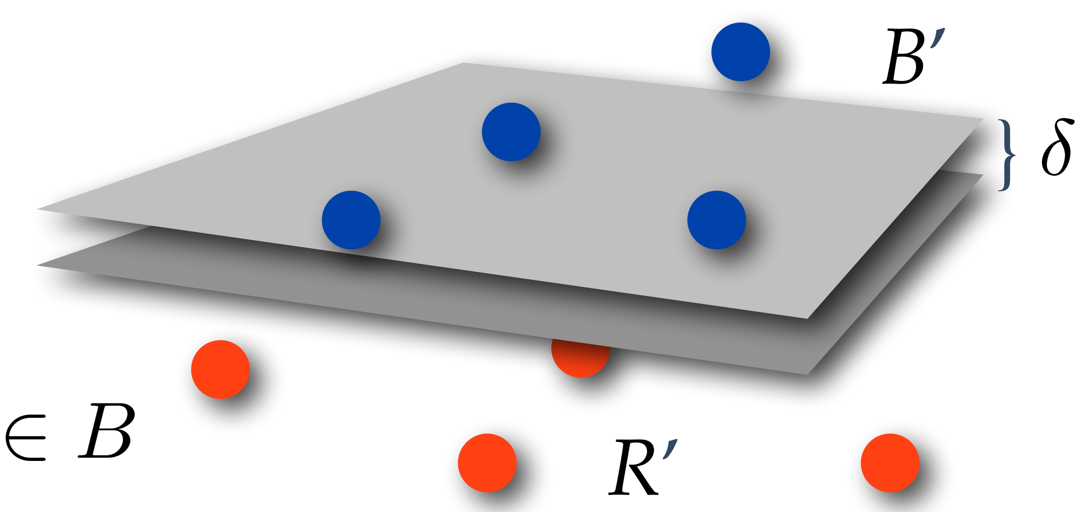
$(3, 6) \in R$

These are linear constraints!

Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?
- This can be solved with linear programming!
- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



$$x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B$$

$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

$$(3, 6) \in R \longrightarrow 3^2 + 6^2 \leq 3\alpha + 6\beta + \gamma$$

These are linear constraints!

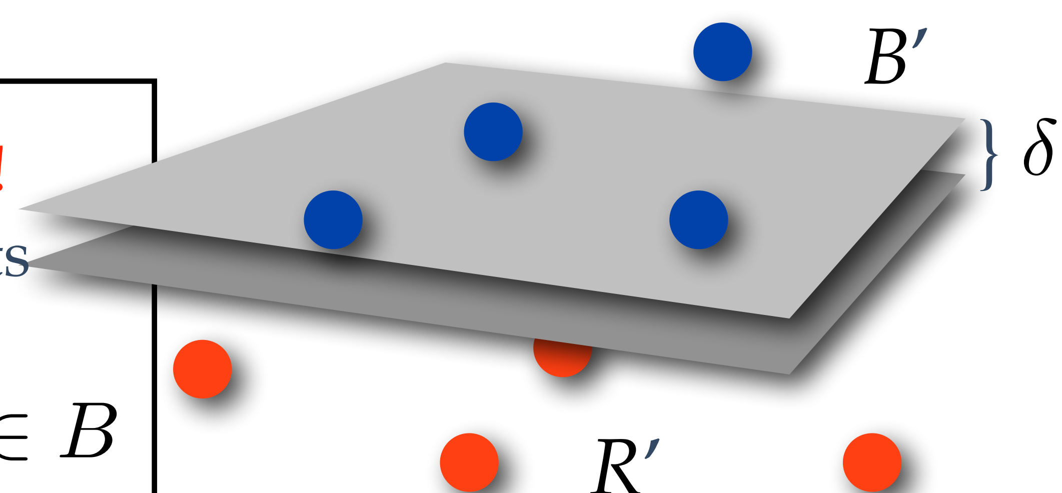
Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?

- This can be solved with linear programming!

- Find α, β, γ, $\delta > 0$ such that...

plane: $z = \alpha x + \beta y + \gamma$



maximize δ

4 variables!!

subject to

$|B| + |R|$ constraints

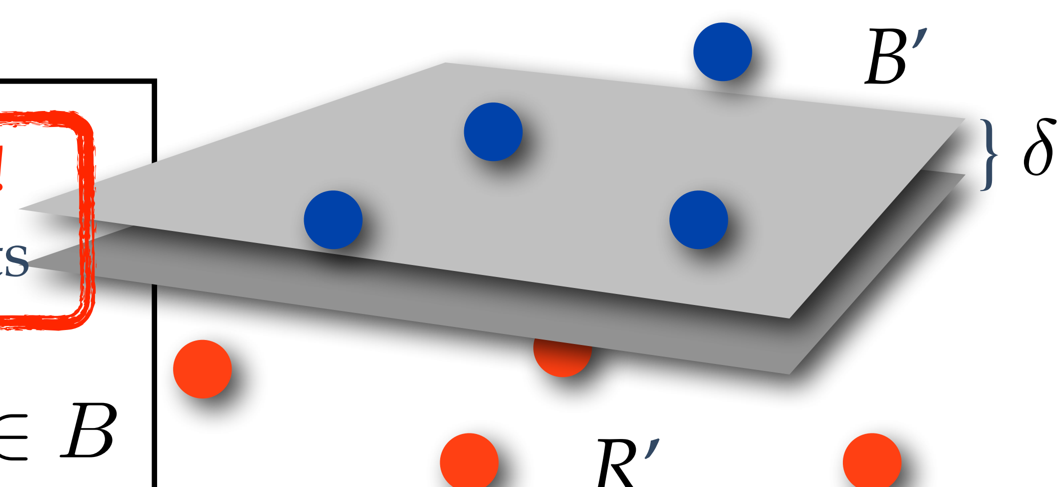
$$x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B$$

$$x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R$$

Linear Programming Application I: Cancer Therapy

- **The geometric problem (lifted space):** Given the lifted sets R' and B' in space, is there a plane that has R' below/on it and B' above?
- This can be solved with linear programming!
- Find α, β, γ, δ > 0 such that...

plane: $z = \alpha x + \beta y + \gamma$



$$
\begin{aligned}
&\text{maximize } \delta \\
&\text{subject to} \\
&\quad x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B \\
&\quad x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R
\end{aligned}
$$

4 variables!!

|B|+|R| constraints

Linear Programming Application I: Cancer Therapy

- **Fact:** Exposure is possible if and only if the following linear program has positive value.

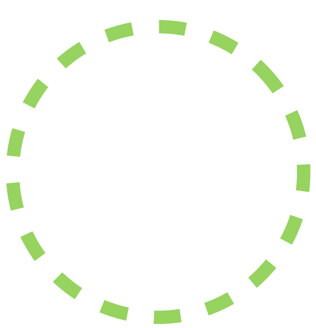
$$\boxed{\begin{array}{l} \text{maximize } \delta \\ \text{subject to} \\ \quad x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B \\ \quad x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R \end{array}}$$

Linear Programming Application I: Cancer Therapy

- **Fact:** Exposure is possible if and only if the following linear program has positive value.

$$
\boxed{
\begin{aligned}
&\text{maximize } \delta \\
&\text{subject to} \\
&\quad x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B \\
&\quad x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R
\end{aligned}
}
$$

- **Reconstructing the exposure from an optimal solution** (α, β, γ, δ):



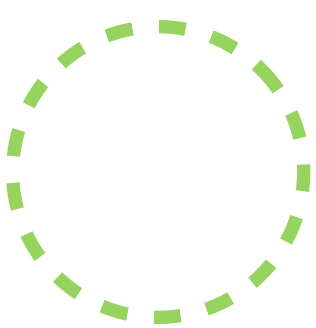
$$= \{(x, y) : x^2 + y^2 = \alpha x + \beta y + \gamma\}$$

Linear Programming Application I: Cancer Therapy

- **Fact:** Exposure is possible if and only if the following linear program has positive value.

$$\boxed{\begin{aligned} &\text{maximize } \delta \\ &\text{subject to} \\ &\quad x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B \\ &\quad x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R \end{aligned}}$$

- **Reconstructing the exposure from an optimal solution** (α, β, γ, δ):



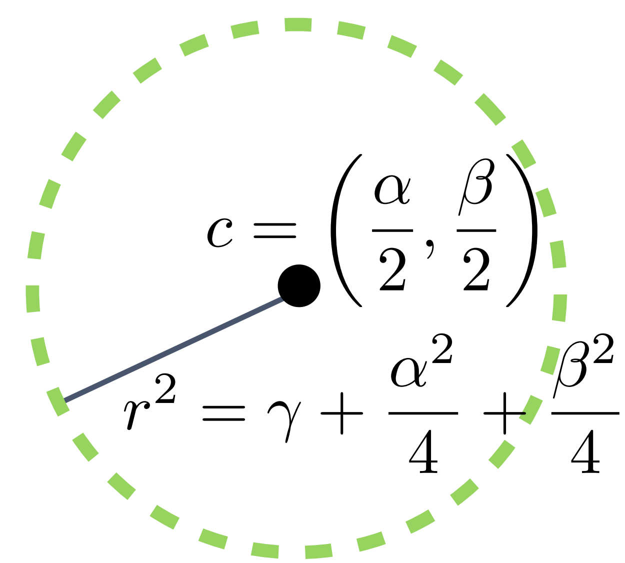
$$= \{(x, y) : x^2 + y^2 = \alpha x + \beta y + \gamma\}$$

$$= \{(x, y) : (x - \frac{\alpha}{2})^2 + (y - \frac{\beta}{2})^2 = \gamma + \frac{\alpha^2}{4} + \frac{\beta^2}{4}\}$$

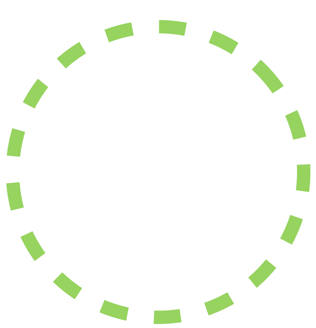
Linear Programming Application I: Cancer Therapy

- **Fact:** Exposure is possible if and only if the following linear program has positive value.

$$
\boxed{\begin{aligned}
&\text{maximize } \delta \\
&\text{subject to} \\
&\quad x^2 + y^2 \geq \alpha x + \beta y + \gamma + \delta, \quad (x, y) \in B \\
&\quad x^2 + y^2 \leq \alpha x + \beta y + \gamma, \quad (x, y) \in R
\end{aligned}}
$$



- **Reconstructing the exposure from an optimal solution** (α, β, γ, δ):



$$= \{(x, y) : x^2 + y^2 = \alpha x + \beta y + \gamma\}$$

$$= \{(x, y) : (x - \frac{\alpha}{2})^2 + (y - \frac{\beta}{2})^2 = \gamma + \frac{\alpha^2}{4} + \frac{\beta^2}{4}\}$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL:**

$$\begin{array}{llcll} \text{minimize} & -\delta & & & \\ \text{subject to} & x^2+y^2 & \geq & \alpha x+\beta y+\gamma+\delta, & (x,y)\in B \\ & x^2+y^2 & \leq & \alpha x+\beta y+\gamma, & (x,y)\in R \\ & \delta & \leq & 1 & \end{array}$$

Avoids unbounded program

maximize $c^T x \rightarrow$ minimize $-c^T x$ and negate resulting value

$$\begin{array}{llll} \text{minimize} & -\delta & & \\ \text{subject to} & x^2+y^2 & \geq & \alpha x + \beta y + \gamma + \delta, \quad (x,y) \in B \\ & x^2+y^2 & \leq & \alpha x + \beta y + \gamma, \quad (x,y) \in R \\ & \delta & \leq & 1 \end{array}$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$
\begin{array}{lrcll}
\text{minimize} & -\delta & & & \\
\text{subject to} & x^2+y^2 & \geq & \alpha x + \beta y + \gamma + \delta, & (x,y) \in B \\
 & \boxed{x^2+y^2 \leq \alpha x + \beta y + \gamma,} & & & (x,y) \in R \\
 & \delta & \leq & 1 &
\end{array}
$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$\begin{array}{llcll} \text{minimize} & & -\delta & & \\ \text{subject to} & \boxed{x^2 + y^2 & \geq & \alpha x + \beta y + \gamma + \delta,} & (x,y) \in B \\ & x^2 + y^2 & \leq & \alpha x + \beta y + \gamma, & (x,y) \in R \\ & \delta & \leq & 1 & \end{array}$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$
\begin{array}{llcll}
\text{minimize} & & \boxed{-\delta} & & \\
\text{subject to} & x^2+y^2 & \geq & \alpha x+\beta y+\gamma+\delta, & (x,y)\in B \\
& x^2+y^2 & \leq & \alpha x+\beta y+\gamma, & (x,y)\in R \\
& \delta & \leq & 1 &
\end{array}
$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$
\begin{array}{llcll}
\text{minimize} & -\delta & & & \\
\text{subject to} & x^2+y^2 & \geq & \alpha x+\beta y+\gamma+\delta, & (x,y)\in B \\
& x^2+y^2 & \leq & \alpha x+\beta y+\gamma, & (x,y)\in R \\
& \delta & \leq & 1 &
\end{array}
$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$
\begin{array}{llcll}
\text{minimize} & -\delta & & & \\
\text{subject to} & x^2 + y^2 & \geq & \alpha x + \beta y + \gamma + \delta, & (x,y) \in B \\
& x^2 + y^2 & \leq & \alpha x + \beta y + \gamma, & (x,y) \in R \\
& \delta & \leq & 1 &
\end{array}
$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Setup and Solve** (Preamble as before)

```
int main() {
  // by default, we have an LP with Ax <= b and no bounds for
  // the four variables alpha, beta, gamma, delta
  Program lp (CGAL::SMALLER, false, 0, false, 0);
  const int alpha = 0;
  const int beta  = 1;
  const int gamma = 2;
  const int delta = 3;

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

  // read the red points (cancer cells)
  for (int i=0; i<m; ++i) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point inside/on circle:
    //   -alpha x - beta y - gamma <= -x^2 - y^2
    lp.set_a (alpha, i, -x);
    lp.set_a (beta,  i, -y);
    lp.set_a (gamma, i, -1);
    lp.set_b (       i, -x*x - y*y);
  }
```

```
  // read the blue points (healthy cells)
  for (int j=0; j<n; ++j) {
    int x; std::cin >> x;
    int y; std::cin >> y;
    // set up <= constraint for point outside circle:
    //   alpha x + beta y + gamma + delta <= x^2 + y^2
    lp.set_a (alpha, m+j, x);
    lp.set_a (beta,  m+j, y);
    lp.set_a (gamma, m+j, 1);
    lp.set_a (delta, m+j, 1);
    lp.set_b (       m+j, x*x + y*y);
  }

  // objective function: -delta (the solver minimizes)
  lp.set_c(delta, -1);

  // enforce a bounded problem:
  lp.set_u (delta, 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));
```

$$
\begin{array}{llll}
\text{minimize} & -\delta & & \\
\text{subject to} & x^2+y^2 & \geq & \alpha x+\beta y+\gamma+\delta, \qquad (x,y)\in B \\
 & x^2+y^2 & \leq & \alpha x+\beta y+\gamma, \qquad (x,y)\in R \\
 & \delta & \leq & 1
\end{array}
$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Output**

negate resulting value!

```
  // output exposure center and radius, if they exist
  if (s.is_optimal() && (s.objective_value() < 0)) {
    // *opt := alpha, *(opt+1) := beta, *(opt+2) := gamma
    CGAL::Quadratic_program_solution<ET>::Variable_value_iterator
      opt = s.variable_values_begin();
    CGAL::Quotient<ET> alpha_opt = *opt +alpha;// +0
    CGAL::Quotient<ET> beta_opt = *(opt+beta);// +1
    CGAL::Quotient<ET> gamma_opt = *(opt+gamma);// +2
    std::cout << "There is a valid exposure:\n";
    std::cout << " Center = ("        // (alpha/2, beta/2)
         << alpha_opt/2 << ", " <<  beta_opt/2
         << ")\n";
    std::cout << " Squared Radius = " // gamma + alpha^2/4 + beta^2/4
         << gamma_opt + alpha_opt*alpha_opt/4 + beta_opt*beta_opt/4 << "\n";
  } else
    std::cout << "There is no valid exposure.";
  std::cout << "\n";
  return 0;
}
```

$$\begin{array}{llcll} \text{minimize} & -\delta & & & \\ \text{subject to} & x^2+y^2 & \geq & \alpha x+\beta y+\gamma+\delta, & (x,y)\in B \\ & x^2+y^2 & \leq & \alpha x+\beta y+\gamma, & (x,y)\in R \\ & \delta & \leq & 1 & \end{array}$$

Linear Programming Application I: Cancer Therapy

- **Implementation in CGAL: Output**

negate resulting value!

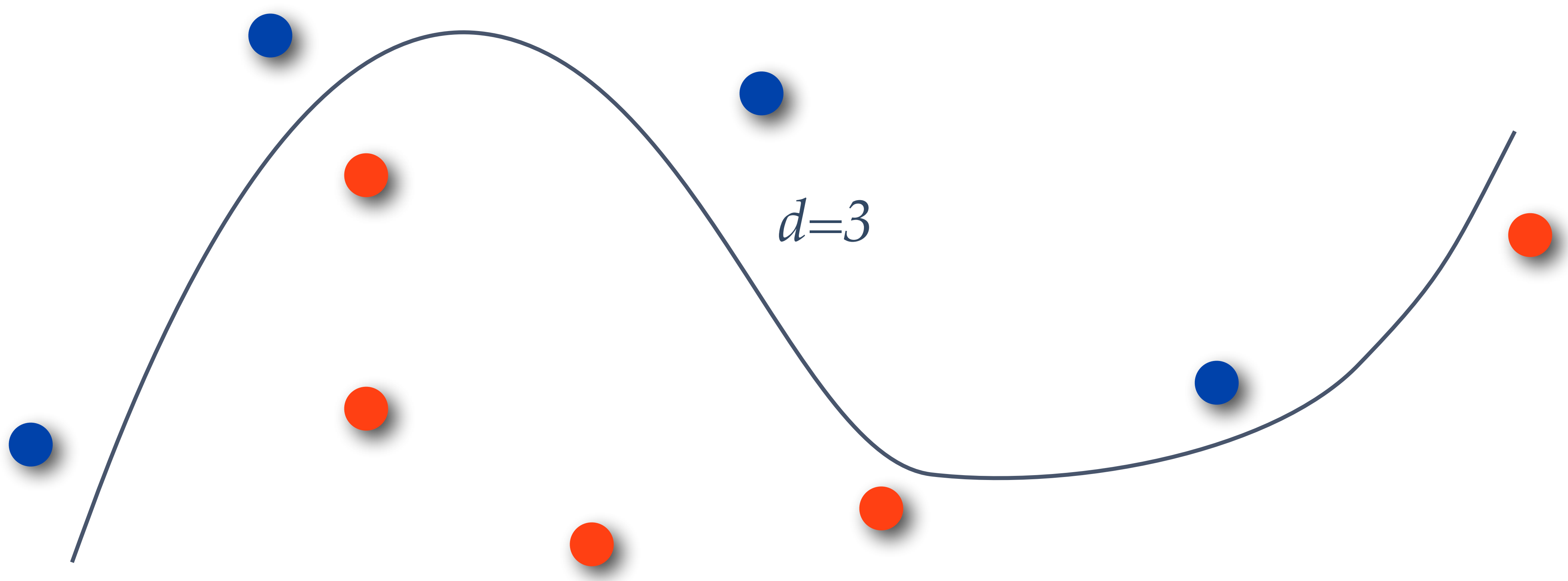
```
  // output exposure center and radius, if they exist
  if (s.is_optimal() && (s.objective_value() < 0)) {
    // *opt := alpha, *(opt+1) := beta, *(opt+2) := gamma
    CGAL::Quadratic_program_solution<ET>::Variable_value_iterator
      opt = s.variable_values_begin();
    CGAL::Quotient<ET> alpha_opt = *opt +alpha;// +0
    CGAL::Quotient<ET> beta_opt = *(opt+beta);// +1
    CGAL::Quotient<ET> gamma_opt = *(opt+gamma);// +2
    std::cout << "There is a valid exposure:\n";
    std::cout << " Center = ("        // (alpha/2, beta/2)
         << alpha_opt/2 << ", " <<  beta_opt/2
         << ")\n";
    std::cout << " Squared Radius = " // gamma + alpha^2/4 + beta^2/4
         << gamma_opt + alpha_opt*alpha_opt/4 + beta_opt*beta_opt/4 << "\n";
  } else
    std::cout << "There is no valid exposure.";
  std::cout << "\n";
  return 0;
}
```

"Pointer" to first variable of optimal solution

The quotient **`*(opt+i)`** is the value of the variable x_i in the optimal solution

Linear Programming Beyond Cancer Therapy

- Given a set of R of red and a set B of blue points, can they be separated by the zero set of a polynomial of degree *d*?



Linear Programming Beyond Cancer Therapy

- Given a set of R of red and a set B of blue points, can they be separated by the zero set of a polynomial of degree d?

- Polynomial of degree 3:

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j$$

Linear Programming Beyond Cancer Therapy

- Given a set of R of red and a set B of blue points, can they be separated by the zero set of a polynomial of degree *d*?

- Polynomial of degree 3:

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j$$

- Linear programming formulation: find *a,b,c,d,e,f,g,h,i,j* such that

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j \leq 0, \quad (x, y) \in B$$
$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j \geq 0, \quad (x, y) \in R$$

Linear Programming Beyond Cancer Therapy

- Given a set of R of red and a set B of blue points, can they be separated by the zero set of a polynomial of degree *d*?

- Polynomial of degree 3:

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j$$

- Linear programming formulation: find *a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*,*j* such that

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j \leq 0, \quad (x,y) \in B$$
$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j \geq 0, \quad (x,y) \in R$$

- This is linear separability in 9-dimensional space, under the generalized lifting map $(x,y) \rightarrow (x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y)$

Linear Programming Further Applications

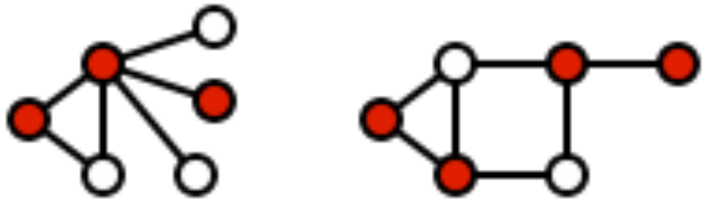
Not necessary for the class, but good to know

- Linear programming relaxations for hard combinatorial problems

Linear Programming Further Applications

Not necessary for the class, but good to know

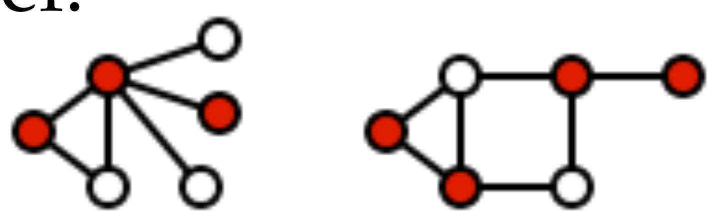
- Linear programming relaxations for hard combinatorial problems
- **Vertex Cover:** Given a graph G=(V,E), find a smallest subset of vertices (a vertex cover) such that every edge is incident to one vertex of the cover.



Linear Programming Further Applications

Not necessary for the class, but good to know

- Linear programming relaxations for hard combinatorial problems
- **Vertex Cover:** Given a graph G=(V,E), find a smallest subset of vertices (a vertex cover) such that every edge is incident to one vertex of the cover.



$x_i \in \{0,1\}$

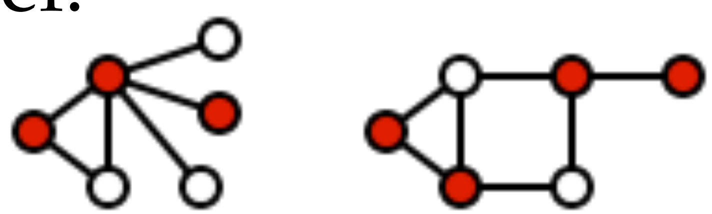
- Formulation as "LP": x_i indicates whether vertex i is in the cover (0: not in the cover, 1: in the cover):

$$
\begin{array}{lrcl}
\text{minimize} & \sum_{i=1}^{n} x_i & & \\
\text{subject to} & x_i + x_j & \geq & 1 \quad \forall \{i,j\} \in E \\
& 0 \leq x_i \leq 1 & & \forall i \in V
\end{array}
$$

Linear Programming Further Applications

Not necessary for the class, but good to know

- Linear programming relaxations for hard combinatorial problems
- **Vertex Cover:** Given a graph G=(V,E), find a smallest subset of vertices (a vertex cover) such that every edge is incident to one vertex of the cover.



- Formulation as "LP": x_i indicates whether vertex i is in the cover (0: not in the cover, 1: in the cover):

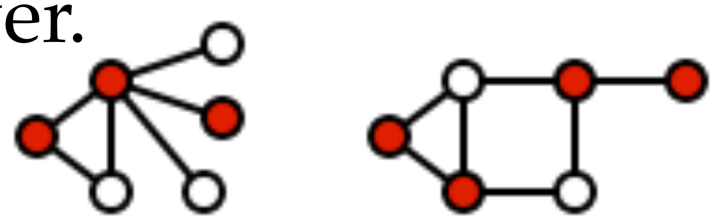
$$\begin{array}{llcll} \text{minimize} & \sum_{i=1}^{n} x_i & & & \\ \text{subject to} & x_i + x_j & \geq & 1 & \forall \{i,j\} \in E \\ & 0 \leq x_i \leq 1 & & & \forall i \in V \\ & x_i \in \{0,1\} & & & \forall i \in V \end{array}$$

not an LP!

Linear Programming Further Applications

Not necessary for the class, but good to know

- **Vertex Cover:** Given a graph G=(V,E), find a smallest subset of vertices (a vertex cover) such that every edge is incident to one vertex of the cover.



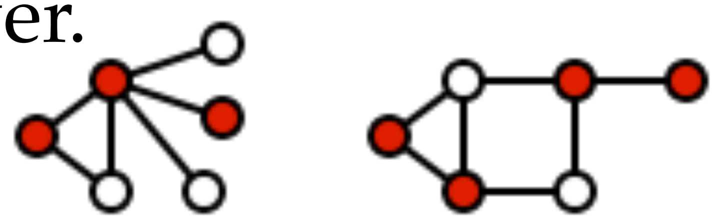
- Let $x^*_1, x^*_2, \ldots, x^*_n$ be an optimal solution of the *LP relaxation*

$$
\begin{array}{lrcl}
\text{minimize} & \sum_{i=1}^{n} x_i & & \\
\text{subject to} & x_i + x_j & \geq & 1 \quad \forall \{i,j\} \in E \\
& 0 \leq x_i \leq 1 & & \quad \forall i \in V
\end{array}
$$

Linear Programming Further Applications

Not necessary for the class, but good to know

- **Vertex Cover:** Given a graph G=(V,E), find a smallest subset of vertices (a vertex cover) such that every edge is incident to one vertex of the cover.



- Let $x^*_1, x^*_2, \ldots, x^*_n$ be an optimal solution of the *LP relaxation*

$$
\begin{array}{llcl}
\text{minimize} & \sum_{i=1}^{n} x_i & & \\
\text{subject to} & x_i + x_j & \geq & 1 \quad \forall \{i,j\} \in E \\
& 0 \leq x_i \leq 1 & & \forall i \in V
\end{array}
$$

- **Theorem:** $C = \{i : x^*_i \geq 1/2\}$ is a vertex cover of size at most 2 opt.

[Ref]

Linear vs. Integer Programming

- Often, applications lead to linear programs with the additional requirement of *integral solutions* (e.g. vertex cover)

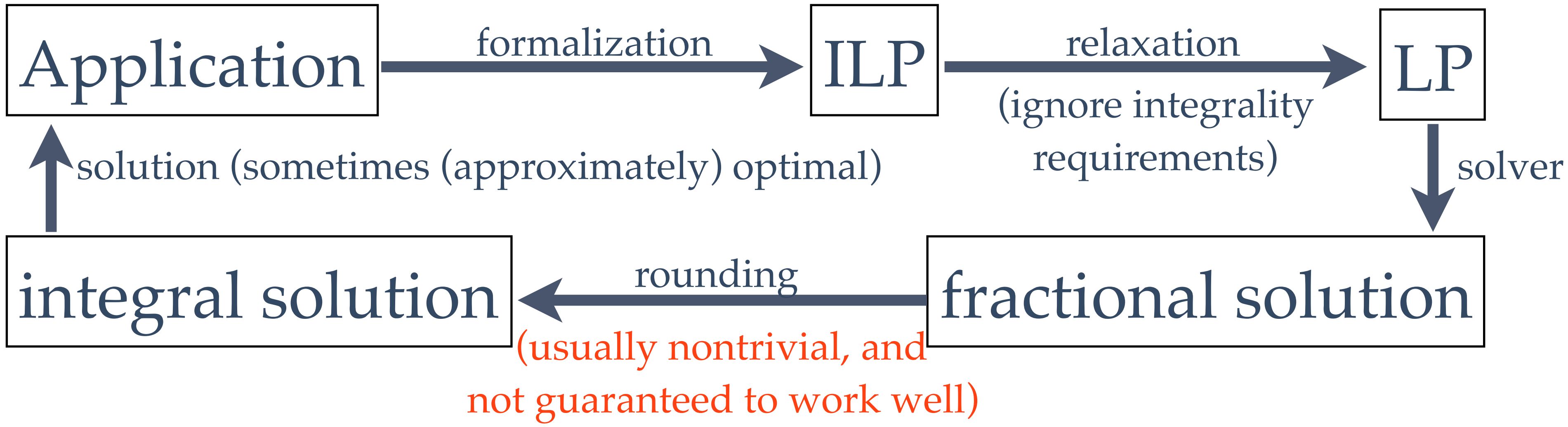
Linear vs. Integer Programming

- Often, applications lead to linear programs with the additional requirement of *integral solutions* (e.g. vertex cover)

- Such programs are called *integer linear programs* (ILP) and are in general much harder to solve than linear programs (NP-hard)

Linear vs. Integer Programming

- Often, applications lead to linear programs with the additional requirement of *integral solutions* (e.g. vertex cover)
- Such programs are called *integer linear programs* (ILP) and are in general much harder to solve than linear programs (NP-hard)
- Typical approach (e.g. vertex cover):



Quadratic Programming (QP)

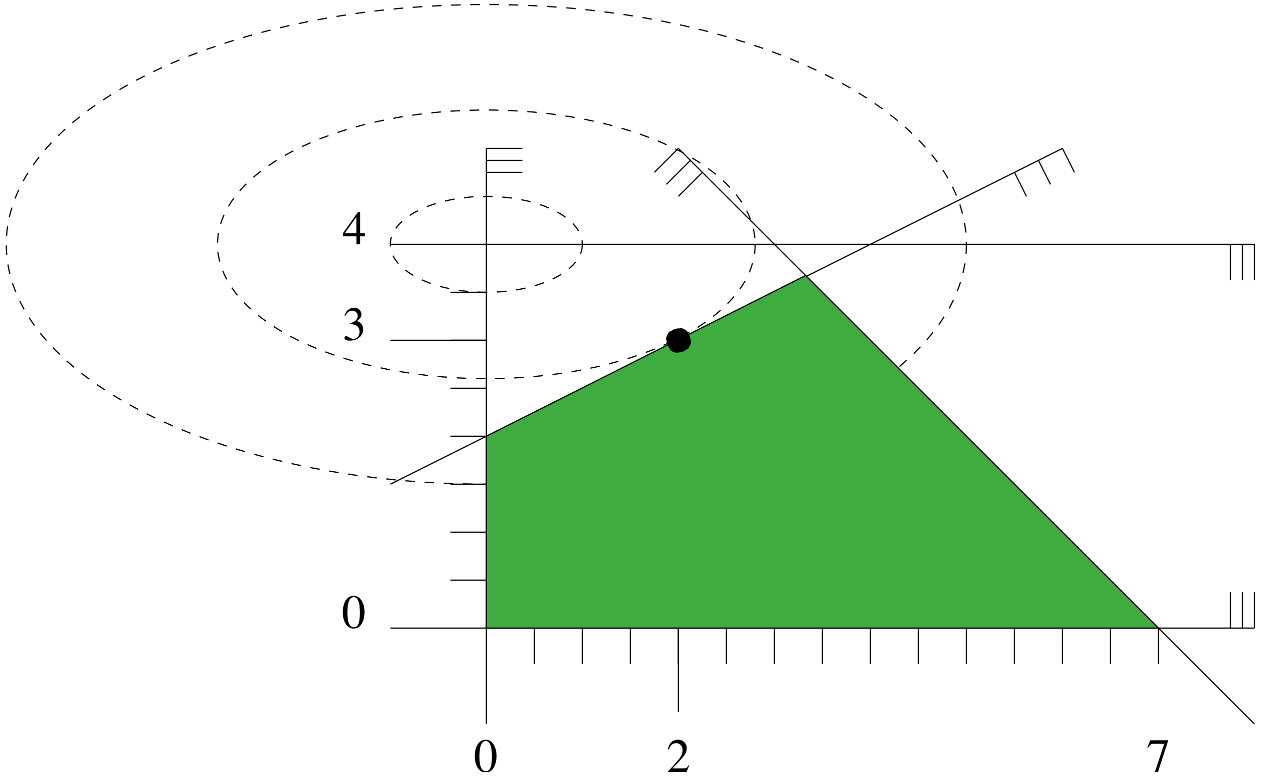
- **Problem:** Minimize a convex quadratic function in n variables subject to m linear (in)equality constraints!

Quadratic Programming

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

- **Example** (n=2, m=5):

$$
\begin{array}{lrcr}
\text{minimize} & x^2 + 4y^2 - 32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
& -x + 2y & \leq & 4 \\
& x & \geq & 0 \\
& y & \geq & 0 \\
& y & \leq & 4
\end{array}
$$



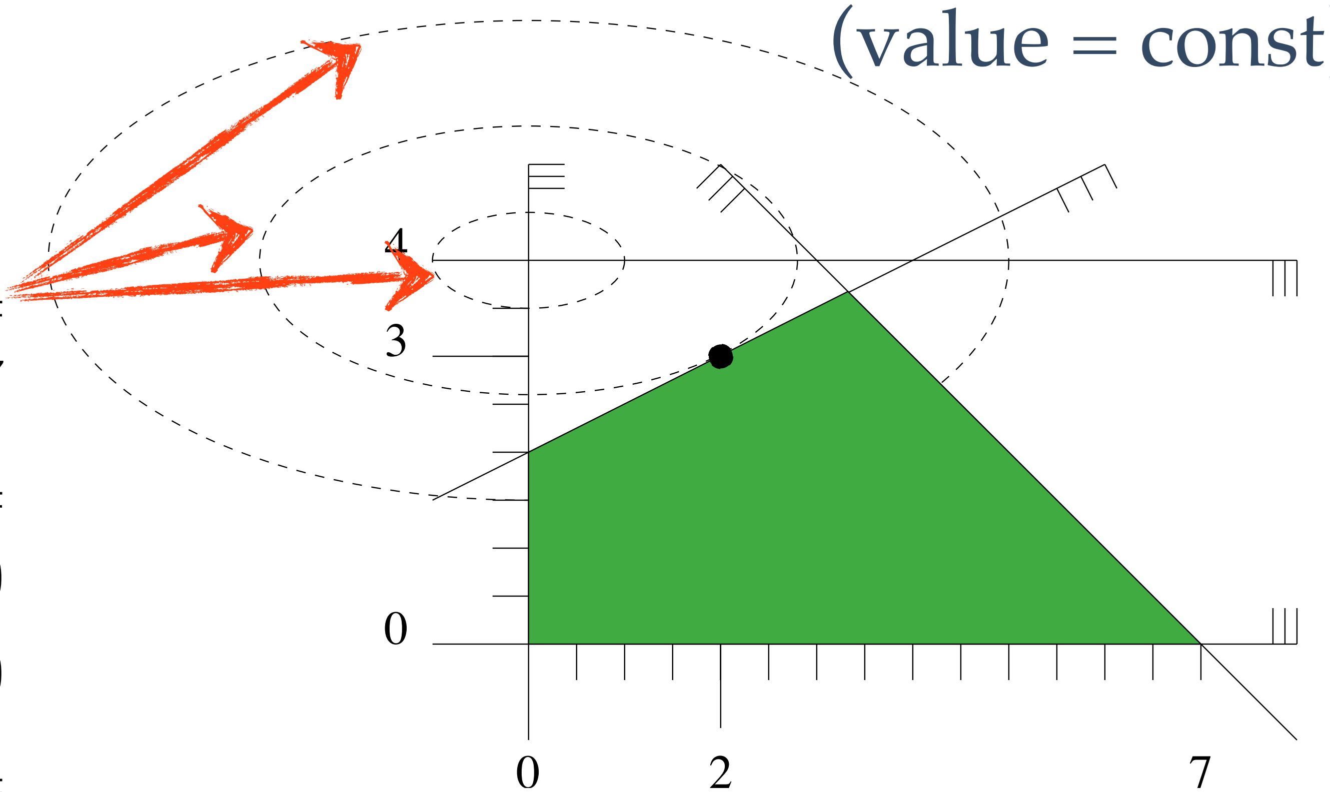
Quadratic Programming

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

- **Example** (n=2, m=5):

$$
\begin{array}{lrcr}
\text{minimize} & x^2 + 4y^2 - 32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
$$

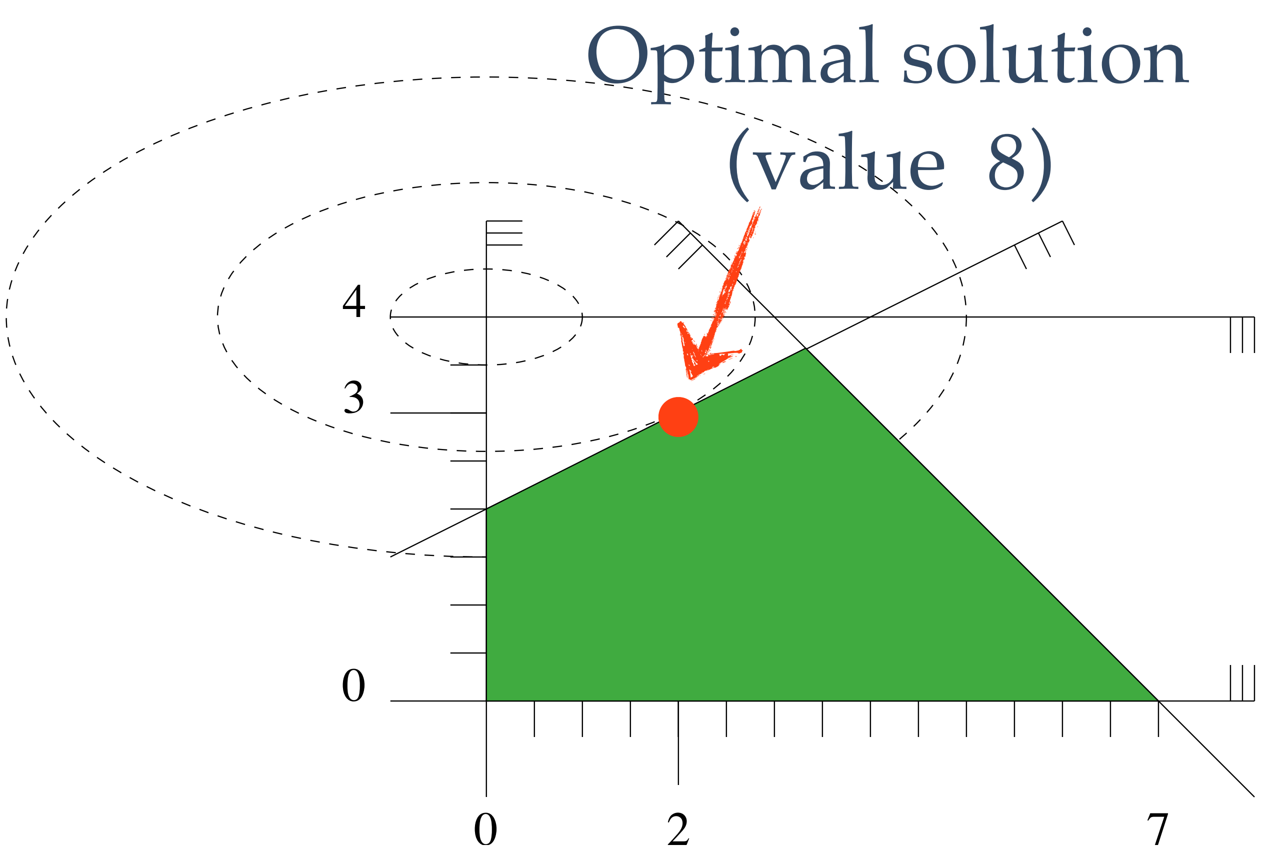
isolines
(value = const)



Quadratic Programming

- **Problem:** Minimize a convex quadratic function in n variables subject to m linear (in)equality constraints!
- **Example** (n=2, m=5):

$$\begin{array}{lrcr} \text{minimize} & x^2 + 4y^2 - 32y + 64 & & \\ \text{subject to} & x + y & \leq & 7 \\ & -x + 2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array}$$



Quadratic Programming ... in CGAL

- **General form of QP in CGAL:**

$$
\begin{array}{ll}
\text{minimize} & x^T D x + c^T x + c_0 \\
\text{subject to} & A x \gtreqless b \\
 & l \leq x \leq u
\end{array}
$$

$(D \in \mathbb{R}^{n \times n}$ symmetric positive semidefinite$)$

Quadratic Programming ... in CGAL

- **General form of QP in CGAL:**

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

$$(D \in \mathbb{R}^{n \times n} \text{ symmetric positive semidefinite})$$

- **Warning:** if D is not positive semidefinite, the quadratic objective function is not convex, and we know not how to solve it.
The CGAL solver might in this case return solutions that are not optimal, or it might crash.

Quadratic Programming ... in CGAL

- **General form of QP in CGAL:**

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

$$(D \in \mathbb{R}^{n \times n} \text{ symmetric positive semidefinite})$$

- **Warning:** if D is not positive semidefinite, the quadratic objective function is not convex, and we know not how to solve it.
The CGAL solver might in this case return solutions that are not optimal, or it might crash.

- **Relax:** In the applications, we know from theory that *D* is "good"

Quadratic Programming ... in CGAL

- **General form of QP in CGAL:**

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

$$(D \in \mathbb{R}^{n \times n} \text{ symmetric positive semidefinite})$$

- **Example:**

$$\begin{array}{lrcr} \text{minimize} & \multicolumn{3}{l}{x^2 + 4y^2 - 32y + 64} \\ \text{subject to} & x + y & \leq & 7 \\ & -x + 2y & \leq & 4 \\ & x & \geq & 0 \\ & y & \geq & 0 \\ & y & \leq & 4 \end{array} \quad \Rightarrow \quad D = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \checkmark$$

Quadratic Programming Application: Low-Risk Investment

- **Problem:** How to invest money such that the expected return is maximized but the risk is minimized?

Quadratic Programming Application: Low-Risk Investment

- **Problem:** How to invest money such that the expected return is maximized but the risk is minimized?

- **Fact:** There is no free lunch (= infinite return with no risk), so we have to accept some risk, and/or live with moderate returns.

Quadratic Programming Application: Low-Risk Investment

- **Problem:** How to invest money such that the expected return is maximized but the risk is minimized?

- **Fact:** There is no free lunch (= infinite return with no risk), so we have to accept some risk, and/or live with moderate returns.

- **Risk-averse strategy:** Maximize the expected return under a given upper bound for the risk!

Quadratic Programming Application: Low-Risk Investment

- **Problem:** How to invest money such that the expected return is maximized but the risk is minimized?

- **Fact:** There is no free lunch (= infinite return with no risk), so we have to accept some risk, and/or live with moderate returns.

- **Risk-averse strategy:** Maximize the expected return under a given upper bound for the risk!

- **Risk-tolerant strategy:** Minimize the risk under a given lower bound for the expected return!

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - *1,2,...,n* (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - $1,2,\ldots,n$ (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - *1,2,...,n* (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i: return rate of investment i (assumed to be a random variable)

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - *1,2,...,n* (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i : return rate of investment i (assumed to be a random variable)
  - r_i : expected return rate of investment i, E [R_i]

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - $1,2,\ldots,n$ (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i : return rate of investment i (assumed to be a random variable)
  - r_i : expected return rate of investment i, $E[R_i]$
  - v_i : variance ("risk") of R_i, $\mathrm{Var}[R_i] := E[(R_i - E[R_i])^2]$

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - *1,2,...,n* (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i : return rate of investment i (assumed to be a random variable)
  - r_i : expected return rate of investment i, E $[R_i]$
  - v_i : variance ("risk") of R_i , Var $[R_i]$:= E $[(R_i - E[R_i])^2]$
  - v_{ij} : covariance ("correlation") of R_i and R_j, E $[(R_i - E[R_i]) (R_j - E[R_j])]$

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - *1,2,...,n* (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i : return rate of investment i (assumed to be a random variable)
  - r_i : expected return rate of investment i, E $[R_i]$
  - v_i : variance ("risk") of R_i, Var $[R_i]$:= E $[(R_i - E[R_i])^2]$
  - v_{ij} : covariance ("correlation") of R_i and R_j, E $[(R_i - E[R_i]) (R_j - E[R_j])]$

Quadratic Programming Application: Low-Risk Investment

- Possible **investments:**
  - $1,2,\ldots,n$ (e.g. 1 = Swatch shares, 2 = Credit Suisse shares,...)
- **Investment Characteristics** (not at all easy to know/estimate)**:**
  - R_i : return rate of investment i (assumed to be a random variable)
  - r_i : expected return rate of investment i, $E[R_i]$
  - v_i : variance ("risk") of R_i, $\mathrm{Var}[R_i] := E[(R_i - E[R_i])^2]$ $\boxed{v_{ii} = v_i}$
  - v_{ij} : covariance ("correlation") of R_i and R_j, $E[(R_i - E[R_i])(R_j - E[R_j])]$

Quadratic Programming Application: Low-Risk Investment

- **Example:** n=2

	r_i
Swatch shares	10% (0.1)
Credit Suisse shares	51% (0.51)

v_{ij}	Swatch shares	Credit Suisse shares
Swatch shares	0.09	-0.05
Credit Suisse shares	-0.05	0.25

Quadratic Programming Application: Low-Risk Investment

- **Example:** n=2

	r_i
Swatch shares	10% (0.1)
Credit Suisse shares	51% (0.51)

v_{ij}	Swatch shares	Credit Suisse shares
Swatch shares	0.09	-0.05
Credit Suisse shares	-0.05	0.25

Negative correlation: if CS does worse than expected, Swatch will probably do better, and vice versa

Quadratic Programming Application: Low-Risk Investment

- **Example:** n=2

	r_i
Swatch shares	10% (0.1)
Credit Suisse shares	51% (0.51)

v_{ij}	Swatch shares	Credit Suisse shares
Swatch shares	0.09	-0.05
Credit Suisse shares	-0.05	0.25

Read as: standard deviation of return rate is $\sqrt{0.25} = 0.5$
(actual return rate could easily be off by 0.5)

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

- **Expected return rate of this strategy:**

$$E[\sum_{i=1}^{n} x_i R_i] = \sum_{i=1}^{n} x_i E[R_i] = \sum_{i=1}^{n} r_i x_i$$

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

- **Expected return rate of this strategy:**

$$E[\sum_{i=1}^{n} x_i R_i] = \sum_{i=1}^{n} x_i E[R_i] = \sum_{i=1}^{n} r_i x_i$$

- **Example:** half the money in Swatch shares, half in Credit Suisse shares; expected return rate is $\frac{1}{2} \cdot 0.1 + \frac{1}{2} \cdot 0.51 = 0.305 = 30.5\%$

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

- **Risk of this strategy:**

Straightforward calculations

$$\mathrm{Var}[\sum_{i=1}^{n} x_i R_i] = \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_i x_j = x^T D x$$

$$D = (v_{ij})_{1 \leq i,j \leq n} \text{is the } \textit{covariance matrix}$$

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

- **Risk of this strategy:**

Straightforward calculations

$$\mathrm{Var}[\sum_{i=1}^{n} x_i R_i] = \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_i x_j = x^T D x$$

$$D = (v_{ij})_{1 \leq i,j \leq n} \text{is the } \textit{covariance matrix}$$

- **Example:** half-half Swatch/CS has risk $\dfrac{0.09 - 2 \cdot 0.05 + 0.25}{4} = 0.06$

Quadratic Programming Application: Low-Risk Investment

- **Investment strategy:**

$$(x_1, x_2, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0 \forall i$$

Meaning: An x_i fraction of your money goes into investment *i*

- **Risk of this strategy:**

Straightforward calculations

$$\mathrm{Var}[\sum_{i=1}^{n} x_i R_i] = \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_i x_j = x^T D x$$

$$D = (v_{ij})_{1 \leq i,j \leq n} \text{is the } \textit{covariance matrix}$$

less than each individual risk!

- **Example:** half-half Swatch/CS has risk $\dfrac{0.09 - 2 \cdot 0.05 + 0.25}{4} = 0.06$

Quadratic Programming Application: Low-Risk Investment

- **The risk-tolerant case:** Find the investment strategy with lowest risk that guarantees expected return rate ρ at least!

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_i x_j \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \ge \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \ge 0, \quad i = 1, \ldots, n \end{aligned}$$

risk

expected return rate

strategy

Quadratic Programming Application: Low-Risk Investment

- **The risk-tolerant case:** Find the investment strategy with lowest risk that guarantees expected return rate ρ at least!

$$
\begin{aligned}
\text{minimize} \quad & \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j}_{\text{risk}} \\
\text{subject to} \quad & \underbrace{\sum_{i=1}^{n} r_i x_i}_{\text{expected return rate}} \geq \rho \\
& \sum_{i=1}^{n} x_i = 1 \\
& x_i \geq 0, \quad i = 1, \ldots, n
\end{aligned}
$$

strategy

Fact: A covariance matrix is positive semidefinite, so this is indeed a convex QP.

Quadratic Programming Application: Low-Risk Investment

- **The risk-tolerant case:** Find the investment strategy with lowest risk that guarantees expected return rate ρ at least!

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j \longleftarrow \boxed{\text{risk}} \\
\text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \qquad \boxed{\text{expected return rate}} \\
& \sum_{i=1}^{n} x_i = 1 \\
\boxed{\text{strategy}} \quad & x_i \geq 0, \quad i = 1, \ldots, n
\end{aligned}
$$

Fact: A covariance matrix is positive semidefinite, so this is indeed a convex QP.

- **Example:** $\rho = 0.4$: 26.8% Swatch, 73.2% Credit Suisse; risk $= 0.121$

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^TDx \\ \text{subject to} \quad & \sum_{i=1}^{n} r_ix_i \ge \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \ge 0, \quad i = 1,\dots,n \end{aligned}$$

- **Preamble:** This time, the input is rational…

Gnu Multi-precision Library (GMP)

```
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpq.h>

// choose exact Rational type
typedef CGAL::Gmpq ET;


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

Low-Risk Investment Example... in CGAL

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j = x^T D x \\
\text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\
& \sum_{i=1}^{n} x_i = 1 \\
& x_i \geq 0, \quad i = 1, \ldots, n
\end{aligned}
$$

- **Input:** Desired expected return

```
int main() {
  // read minimum expected return rate
  std::cout << "What is your desired expected return rate? ";
  double rho; std::cin >> rho;
```

for example, $0.4 \approx 40\%$

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij} x_i x_j = x^T D x \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

- **Setup:** Make sure to enter matrix *2D* (customary in QP solvers)!

```
// by default, we have a nonnegative QP with Ax >= b
Program qp (CGAL::LARGER, true, 0, false, 0);

// now set the non-default entries:
const int sw = 0;
const int cs = 1;

// constraint on expected return: 0.1 sw + 0.51 cs >= rho
qp.set_a(sw, 0, ET(1)/10);
qp.set_a(cs, 0, ET(51)/100);
qp.set_b(    0, rho);

// strategy constraint: sw + cs = 1
qp.set_a(sw, 1, 1);
qp.set_a(cs, 1, 1);
qp.set_b(    1, 1);
qp.set_r(    1, CGAL::EQUAL); // override default >=

// objective function: 0.09 sw^2 - 0.05 sw cs - 0.05 cs sw + 0.25 cs^2
// we need to specify the entries of the symmetric matrix 2D, on and below the diagonal
qp.set_d(sw, sw,  ET(18)/100); //  0.09 sw^2
qp.set_d(cs, sw, -ET(1)/10);   // -0.05 cs sw
qp.set_d(cs, cs,  ET(1)/2);    //  0.25 cs^2
```

Low-Risk Investment Example… in CGAL

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^TDx \\
\text{subject to} \quad & \boxed{\sum_{i=1}^{n} r_ix_i \geq \rho} \\
& \sum_{i=1}^{n} x_i = 1 \\
& x_i \geq 0, \quad i = 1, \ldots, n
\end{aligned}$$

- **Setup:** Make sure to enter matrix *2D* (customary in QP solvers)!

```
// by default, we have a nonnegative QP with Ax >= b
Program qp (CGAL::LARGER, true, 0, false, 0);

// now set the non-default entries:
const int sw = 0;
const int cs = 1;

// constraint on expected return: 0.1 sw + 0.51 cs >= rho
qp.set_a(sw, 0, ET(1)/10);
qp.set_a(cs, 0, ET(51)/100);
qp.set_b(    0, rho);

// strategy constraint: sw + cs = 1
qp.set_a(sw, 1, 1);
qp.set_a(cs, 1, 1);
qp.set_b(    1, 1);
qp.set_r(    1, CGAL::EQUAL); // override default >=

// objective function: 0.09 sw^2 - 0.05 sw cs - 0.05 cs sw + 0.25 cs^2
// we need to specify the entries of the symmetric matrix 2D, on and below the diagonal
qp.set_d(sw, sw,  ET(18)/100); //  0.09 sw^2
qp.set_d(cs, sw, -ET(1)/10);   // -0.05 cs sw
qp.set_d(cs, cs,  ET(1)/2);    //  0.25 cs^2
```

Low-Risk Investment Example… in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j = x^T D x \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\ & \boxed{\sum_{i=1}^{n} x_i = 1} \\ & x_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

- **Setup:** Make sure to enter matrix *2D* (customary in QP solvers)!

```
// by default, we have a nonnegative QP with Ax >= b
Program qp (CGAL::LARGER, true, 0, false, 0);

// now set the non-default entries:
const int sw = 0;
const int cs = 1;

// constraint on expected return: 0.1 sw + 0.51 cs >= rho
qp.set_a(sw, 0, ET(1)/10);
qp.set_a(cs, 0, ET(51)/100);
qp.set_b(    0, rho);

// strategy constraint: sw + cs = 1
qp.set_a(sw, 1, 1);
qp.set_a(cs, 1, 1);
qp.set_b(    1, 1);
qp.set_r(    1, CGAL::EQUAL); // override default >=

// objective function: 0.09 sw^2 - 0.05 sw cs - 0.05 cs sw + 0.25 cs^2
// we need to specify the entries of the symmetric matrix 2D, on and below the diagonal
qp.set_d(sw, sw,  ET(18)/100); //  0.09 sw^2
qp.set_d(cs, sw, -ET(1)/10);   // -0.05 cs sw
qp.set_d(cs, cs,  ET(1)/2);    //  0.25 cs^2
```

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j = x^T D x \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \dots, n \end{aligned}$$

- **Setup:** Make sure to enter matrix *2D* (customary in QP solvers)!

```
// by default, we have a nonnegative QP with Ax >= b
Program qp (CGAL::LARGER, true, 0, false, 0);

// now set the non-default entries:
const int sw = 0;
const int cs = 1;

// constraint on expected return: 0.1 sw + 0.51 cs >= rho
qp.set_a(sw, 0, ET(1)/10);
qp.set_a(cs, 0, ET(51)/100);
qp.set_b(    0, rho);

// strategy constraint: sw + cs = 1
qp.set_a(sw, 1, 1);
qp.set_a(cs, 1, 1);
qp.set_b(    1, 1);
qp.set_r(    1, CGAL::EQUAL); // override default >=

// objective function: 0.09 sw^2 - 0.05 sw cs - 0.05 cs sw + 0.25 cs^2
// we need to specify the entries of the symmetric matrix 2D, on and below the diagonal
qp.set_d(sw, sw,  ET(18)/100); //  0.09 sw^2
qp.set_d(cs, sw, -ET(1)/10);   // -0.05 cs sw
qp.set_d(cs, cs,  ET(1)/2);    //  0.25 cs^2
```

j ≤ i in **`set_d(i, j)`**

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^TDx \\ \text{subject to} \quad & \sum_{i=1}^{n} r_ix_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

- **Solve:** ...as nonnegative quadratic program (a little faster)

```
 // solve the program, using ET as the exact type
assert(qp.is_nonnegative());
Solution s = CGAL::solve_nonnegative_quadratic_program(qp, ET());
assert (s.solves_quadratic_program(qp));
```

Independent verifications.
Only do something in Debug mode.

Low-Risk Investment Example… in CGAL

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^TDx \\
\text{subject to} \quad & \sum_{i=1}^{n} r_ix_i \geq \rho \\
& \sum_{i=1}^{n} x_i = 1 \\
& x_i \geq 0, \quad i = 1, \ldots, n
\end{aligned}
$$

- **Solve:** ...as nonnegative quadratic program (a little faster)

```
 // solve the program, using ET as the exact type
assert(qp.is_nonnegative());
Solution s = CGAL::solve_nonnegative_quadratic_program(qp, ET());
assert (s.solves_quadratic_program(qp));
```

Independent verifications.
Only do something in Debug mode.

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j = x^T D x \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

- **Output:** query solution status; if feasible, output strategy / risk

```
// output
  if (s.status() == CGAL::QP_INFEASIBLE) {
    std::cout << "Expected return rate " << rho << " cannot be achieved.\n";
  } else {
    assert (s.status() == CGAL::QP_OPTIMAL);
    Solution::Variable_value_iterator opt = s.variable_values_begin();
    ET sw_ratio = opt->numerator() / opt->denominator();
    ET cs_ratio = (opt+1)->numerator() / (opt+1)->denominator();
    ET risk = s.objective_value().numerator() / s.objective_value().denominator();
    double sw_percent = ceil_to_double(100 * *opt);
    std::cout << "Minimum risk investment strategy:\n"
          << sw_ratio << " ~ " << sw_percent << "%" << " Swatch\n"
          << cs_ratio << " ~ " << 100-sw_percent << "%" << " CS\n"
          << "Risk = " << risk
          << " ~ ." << ceil_to_double(100 * s.objective_value())
          << "\n";
  }
```

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij} x_i x_j = x^T D x \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n \end{aligned}$$

- **Output:** query solution status; if feasible, output strategy / risk

```
// output
if (s.status() == CGAL::QP_INFEASIBLE) {
  std::cout << "Expected return rate " << rho << " cannot be achieved.\n";
} else {
  assert (s.status() == CGAL::QP_OPTIMAL);
  Solution::Variable_value_iterator opt = s.variable_values_begin();
  ET sw_ratio = opt->numerator() / opt->denominator();
  ET cs_ratio = (opt+1)->numerator() / (opt+1)->denominator();
  ET risk = s.objective_value().numerator() / s.objective_value().denominator();
  double sw_percent = ceil_to_double(100 * *opt);
  std::cout << "Minimum risk investment strategy:\n"
        << sw_ratio << " ~ " << sw_percent << "%" << " Swatch\n"
        << cs_ratio << " ~ " << 100-sw_percent << "%" << " CS\n"
        << "Risk = " << risk
        << " ~ ." << ceil_to_double(100 * s.objective_value())
        << "\n";
}
```

Low-Risk Investment Example... in CGAL

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^TDx \\ \text{subject to} \quad & \sum_{i=1}^{n} r_ix_i \geq \rho \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1,\ldots,n \end{aligned}$$

- **Output:** query solution status; if feasible, output strategy / risk

```
// output
  if (s.status() == CGAL::QP_INFEASIBLE) {
    std::cout << "Expected return rate " << rho << " cannot be achieved.\n";
  } else {
    assert (s.status() == CGAL::QP_OPTIMAL);
    Solution::Variable_value_iterator opt = s.variable_values_begin();
    ET sw_ratio = opt->numerator() / opt->denominator();
    ET cs_ratio = (opt+1)->numerator() / (opt+1)->denominator();
    ET risk = s.objective_value().numerator() / s.objective_value().denominator();
    double sw_percent = ceil_to_double(100 * *opt);
    std::cout << "Minimum risk investment strategy:\n"
          << sw_ratio << " ~ " << sw_percent << "%" << " Swatch\n"
          << cs_ratio << " ~ " << 100-sw_percent << "%" << " CS\n"
          << "Risk = " << risk
          << " ~ ." << ceil_to_double(100 * s.objective_value())
          << "\n";
  }
```

Low-Risk Investment Example... in CGAL

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_ix_j = x^T D x \\
\text{subject to} \quad & \sum_{i=1}^{n} r_i x_i \geq \rho \\
& \sum_{i=1}^{n} x_i = 1 \\
& x_i \geq 0, \quad i = 1, \dots, n
\end{aligned}
$$

- **Output:** query solution status; if feasible, output strategy / risk

```
// output
  if (s.status() == CGAL::QP_INFEASIBLE) {
    std::cout << "Expected return rate " << rho << " cannot be achieved.\n";
  } else {
    assert (s.status() == CGAL::QP_OPTIMAL);
    Solution::Variable_value_iterator opt = s.variable_values_begin();
    ET sw_ratio = opt->numerator() / opt->denominator();
    ET cs_ratio = (opt+1)->numerator() / (opt+1)->denominator();
    ET risk = s.objective_value().numerator() / s.objective_value().denominator();
    double sw_percent = ceil_to_double(100 * *opt);
    std::cout << "Minimum risk investment strategy:\n"
          << sw_ratio << " ~ " << sw_percent << "%" << " Swatch\n"
          << cs_ratio << " ~ " << 100-sw_percent << "%" << " CS\n"
          << "Risk = " << risk
          << " ~ ." << ceil_to_double(100 * s.objective_value())
          << "\n";
  }
```

Known Bug :=(

- You can't reliably copy or assign instances of the class **CGAL::Quadratic_program_solution<ET>**
- **Workaround 1:** If you want to pass or return such instances to/from a function, pass a pointer to the instance instead!
- **Workaround 2:** If you want to assign a new solution to an existing instance... don't do it!

Sources and Further Reading

- **LP/QP Solver:** Online manual at **`www.cgal.org`**: Online Manual → Combinatorial Algorithms → Linear and Quadratic Programming Solver

- **Cancer Therapy:** J. O'Rourke, S. Kosaraju, and N. Megiddo: Computing Circular Separabiliy, *Discrete & Computational Geometry* 1:105-113 (1986)

- **Low-Risk Investment:** H. Markowitz: Portfolio Selection, *Journal of Finance* 7(1): 77-91 (1952)