Linear Programming

Linear Optimization Problem

$$\begin{array}{rll} \text{minimize} & f(x) & \\ \text{by varying} & x \in \mathbb{R}^n & \\ \text{subject to} & g_i(x) \le 0 & i = 1, \dots, n_g \\ & h_j(x) = 0 & j = 1, \dots, n_h \end{array}$$

We assume that

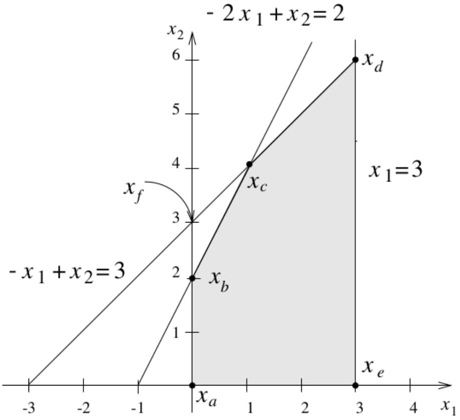
- f is linear, i.e.,

$$f(x) = c^\top x \qquad \text{here } c \in \mathbb{R}^n$$

- each g_i is linear,
- each h_j is linear.

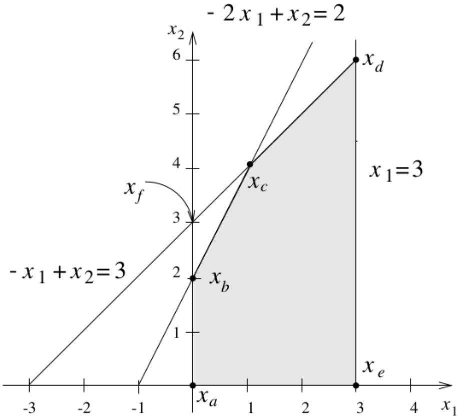
For convenience, in what follows, we also allow constraints of the form $g_i(x) \ge 0$.

Example



$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 - 2 \leq 0 \\
& -x_1 + x_2 - 3 \leq 0 \\
& x_1 - 3 \leq 0 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

Example



The lines define the boundaries of the feasible region

$$-2x_1 + x_2 = 2$$
$$-x_1 + x_2 = 3$$
$$x_1 = 3$$

$$x_1 = 0$$
$$x_2 = 0$$

Standard Form

The *standard form linear program*

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Here

- $x = (x_1, \dots, x_n)^\top \in \mathbb{R}^n$
- $c = (c_1, \dots, c_n)^\top \in \mathbb{R}^n$
- A is an $m \times n$ matrix of elements a_{ij} where $m < n$ and $\text{rank}(A) = m$

  That is, all rows of A are linearly independent.
- $b = (b_1, \dots, b_m)^\top \geq 0$

  $b \geq 0$ means $b_i \geq 0$ for all i.

Standard Form

The *standard form linear program*

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Here

- $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$
- $c = (c_1, \ldots, c_n)^\top \in \mathbb{R}^n$
- A is an $m \times n$ matrix of elements a_{ij} where $m < n$ and $\text{rank}(A) = m$

  That is, all rows of A are linearly independent.
- $b = (b_1, \ldots, b_m)^\top \geq 0$

  $b \geq 0$ means $b_i \geq 0$ for all i.

Every linear optimization problem can be transformed into a standard linear program such that there is a one-to-one correspondence between solutions of the constraints preserving values of the objective.

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

2. Transform every $g_i(x) \leq 0$ to $g_i(x) + s_i = 0, s_i \geq 0$. Here s_i are new variables (*slack variables*).

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

2. Transform every $g_i(x) \leq 0$ to $g_i(x) + s_i = 0, s_i \geq 0$. Here s_i are new variables (*slack variables*).

3. Move all constant terms to the right side of the constraints.

   Now we have constraints of the form $Ax = b, x \geq 0$.

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

2. Transform every $g_i(x) \leq 0$ to $g_i(x) + s_i = 0, s_i \geq 0$. Here s_i are new variables (*slack variables*).

3. Move all constant terms to the right side of the constraints.

   Now we have constraints of the form $Ax = b, x \geq 0$.

4. Remove linearly dependent equations from $Ax = b$.

   This step does not alter the set of solutions.

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

2. Transform every $g_i(x) \leq 0$ to $g_i(x) + s_i = 0, s_i \geq 0$. Here s_i are new variables (*slack variables*).

3. Move all constant terms to the right side of the constraints.

   Now we have constraints of the form $Ax = b, x \geq 0$.

4. Remove linearly dependent equations from $Ax = b$.

   This step does not alter the set of solutions.

5. If $m \geq n$, the constraints either have a unique or no solution. Neither of the cases is interesting for optimization. Hence, $m < n$.

Transformation to Standard Form

1. For every variable x_i introduce new variables x_i', x_i'', replace every occurrence of x_i with $x_i' - x_i''$, and introduce constraints $x_i', x_i'' \geq 0$.

   Note that if a constraint is in the form $x_i + \zeta \geq 0$ we may simply replace x_i with $x_i' - \zeta$ and introduce $x_i' \geq 0$.

2. Transform every $g_i(x) \leq 0$ to $g_i(x) + s_i = 0, s_i \geq 0$. Here s_i are new variables (*slack variables*).

3. Move all constant terms to the right side of the constraints.

   Now we have constraints of the form $Ax = b, x \geq 0$.

4. Remove linearly dependent equations from $Ax = b$.

   This step does not alter the set of solutions.

5. If $m \geq n$, the constraints either have a unique or no solution. Neither of the cases is interesting for optimization. Hence, $m < n$.

6. Multiplying equations with $b_i < 0$ by -1 gives $b \geq 0$

Transformation Example

$$
\begin{aligned}
\text{maximize} \quad & z = -5x_1 - 3x_2 \\
\text{subject to} \quad & 3x_1 - 5x_2 - 5 \leq 0 \\
& -4x_1 - 9x_2 + 4 \leq 0
\end{aligned}
$$

Transformation Example

$$
\begin{array}{ll}
\text{maximize} & z = -5x_1 - 3x_2 \\
\text{subject to} & 3x_1 - 5x_2 - 5 \leq 0 \\
& -4x_1 - 9x_2 + 4 \leq 0
\end{array}
$$

Introduce the bounded variables:

$$
\begin{array}{ll}
\text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\
\text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' - 5 \leq 0 \\
& -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + 4 \leq 0 \\
& x_1', x_1'', x_2', x_2'' \geq 0
\end{array}
$$

Transformation Example

$$\begin{aligned} \text{maximize} \quad & z = -5x_1 - 3x_2 \\ \text{subject to} \quad & 3x_1 - 5x_2 - 5 \le 0 \\ & -4x_1 - 9x_2 + 4 \le 0 \end{aligned}$$

Introduce the bounded variables:

$$\begin{aligned} \text{maximize} \quad & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} \quad & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' - 5 \le 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + 4 \le 0 \\ & x_1', x_1'', x_2', x_2'' \ge 0 \end{aligned}$$

Introduce the slack variables:

$$\begin{aligned} \text{maximize} \quad & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} \quad & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 - 5 = 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 + 4 = 0 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \ge 0 \end{aligned}$$

Transformation Example

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 - 5 = 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 + 4 = 0 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Transformation Example

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 - 5 = 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 + 4 = 0 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Move constants to the right:

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 = 5 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 = -4 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Transformation Example

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 - 5 = 0 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 + 4 = 0 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Move constants to the right:

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 = 5 \\ & -4x_1' + 4x_1'' - 9x_2' + 9x_2'' + s_2 = -4 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Check if all equations are linearly independent.

Multiply the last one with -1:

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 = 5 \\ & 4x_1' - 4x_1'' + 9x_2' - 9x_2'' - s_2 = 4 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

Transformation Example

$$\begin{array}{ll} \text{maximize} & z = -5x_1' + 5x_1'' - 3x_2' + 3x_2'' \\ \text{subject to} & 3x_1' - 3x_1'' - 5x_2' + 5x_2'' + s_1 = 5 \\ & 4x_1' - 4x_1'' + 9x_2' - 9x_2'' - s_2 = 4 \\ & x_1', x_1'', x_2', x_2'', s_1, s_2 \geq 0 \end{array}$$

In the standard form:

$$A = \begin{pmatrix} 3 & -3 & -5 & 5 & 1 & 0 \\ 4 & -4 & 9 & -9 & 0 & -1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5, x_6)^\top$$

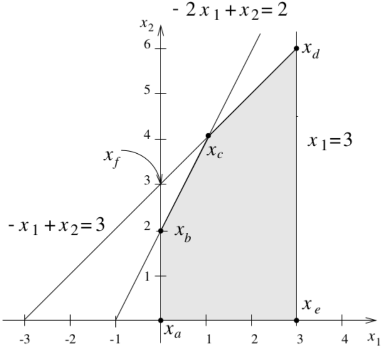
Note that we have renamed the variables.

$$b = (5, 4)^\top$$

$$Ax = b \text{ where } x \geq 0$$

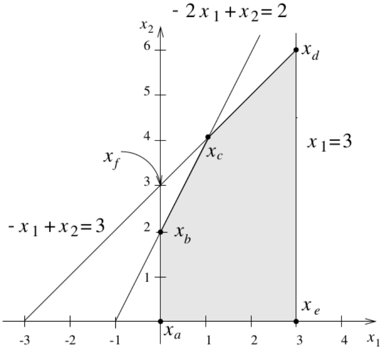
$$c = (-5, 5, -3, 3)^\top$$

Example



$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 - 2 \leq 0 \\
& -x_1 + x_2 - 3 \leq 0 \\
& x_1 - 3 \leq 0 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

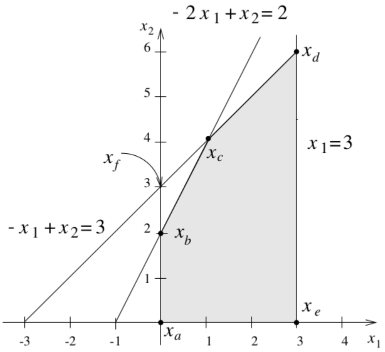
Example



Transform to

$$
\begin{aligned}
\text{minimize} \quad & z = -x_1 - 2x_2 \\
\text{subject to} \quad & -2x_1 + x_2 + s_1 = 2 \\
& -x_1 + x_2 + s_2 = 3 \\
& x_1 + s_3 = 3 \\
& x_1, x_2, s_1, s_2, s_3 \geq 0
\end{aligned}
$$

Example



The standard form:

$$A = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$b = (2, 3, 3)^\top$$

$$Ax = b$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$c = (-1, -2, 0, 0, 0)^\top$$

Assumptions

Consider a linear programming problem in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Assumptions

Consider a linear programming problem in the standard form:

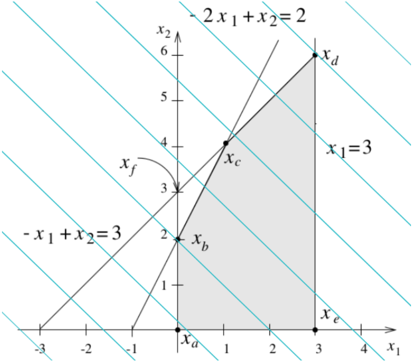
$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

In what follows, we will use the following shorthand: Given two column vectors x, x', we write $[x, x']$ to denote the vector resulting from stacking x on top of x'.

Solutions

There are (typically) infinitely many solutions to the constraints.

Are there some distinguished ones? How do you find minimizers?



Here, the blue lines are contours of $-x_1 - x_2$.

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Denote by N the set of indices of columns not in B.

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Denote by N the set of indices of columns not in B.

Given $x \in \mathbb{R}^n$, we let

- $x_B \in \mathbb{R}^m$ consist of components of x with indices in B
- $x_N \in \mathbb{R}^{n-m}$ consist of components of x with indices in N

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Denote by N the set of indices of columns not in B.

Given $x \in \mathbb{R}^n$, we let

- $x_B \in \mathbb{R}^m$ consist of components of x with indices in B
- $x_N \in \mathbb{R}^{n-m}$ consist of components of x with indices in N

Abusing notation, we denote by B and N the submatrices of A consisting of columns with indices in B and N, resp.

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Denote by N the set of indices of columns not in B.

Given $x \in \mathbb{R}^n$, we let

- $x_B \in \mathbb{R}^m$ consist of components of x with indices in B
- $x_N \in \mathbb{R}^{n-m}$ consist of components of x with indices in N

Abusing notation, we denote by B and N the submatrices of A consisting of columns with indices in B and N, resp.

Basic Solutions

Assume that the matrix A has full row rank (w.l.o.g).

Let B be a set of m indices of columns of A for a linearly independent set. Such a B is called a *basis*.

Denote by N the set of indices of columns not in B.

Given $x \in \mathbb{R}^n$, we let

- $x_B \in \mathbb{R}^m$ consist of components of x with indices in B
- $x_N \in \mathbb{R}^{n-m}$ consist of components of x with indices in N

Abusing notation, we denote by B and N the submatrices of A consisting of columns with indices in B and N, resp.

Definition

Consider $x \in \mathbb{R}^n$ and a basis B, and consider the decomposition of x into $x_B \in \mathbb{R}^m$ and $x_N \in \mathbb{R}^{n-m}$.
Then x is a *basic solution w.r.t. the basis B* if $Ax = b$ and $x_N = 0$.
Components of x_B are *basic variables*.
A basic solution x is *feasible* if $x \geq 0$.

Example (Whiteboard)

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

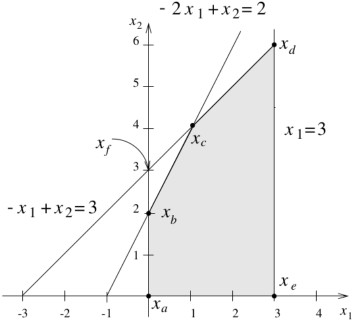
$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (2, 1)^\top$$

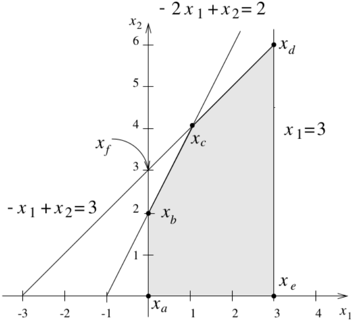
$$Ax = b \text{ where } x \geq 0$$

For now let us ignore the objective function and play with the polyhedron defined by the above inequalities.

$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$

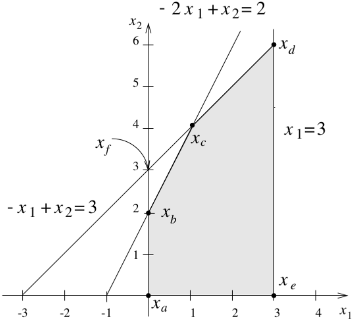


$$\begin{aligned} -2x_1 + x_2 + x_3 &= 2 \\ -x_1 + x_2 + x_4 &= 3 \\ x_1 + x_5 &= 3 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$



$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

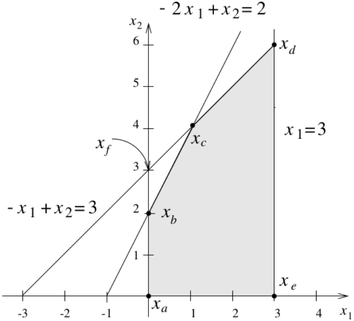
$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$



$$
A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
x = (x_1, x_2, x_3, x_4, x_5)^\top
$$

$$
\begin{aligned}
-2x_1 + x_2 + x_3 &= 2 \\
-x_1 + x_2 + x_4 &= 3 \\
x_1 + x_5 &= 3 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$

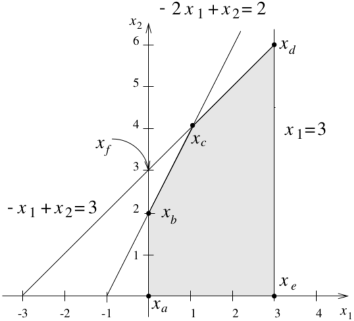


$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

$$\begin{aligned} -2x_1 + x_2 + x_3 &= 2 \\ -x_1 + x_2 + x_4 &= 3 \\ x_1 + x_5 &= 3 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$



$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$b = (2, 3, 3)^\top$$

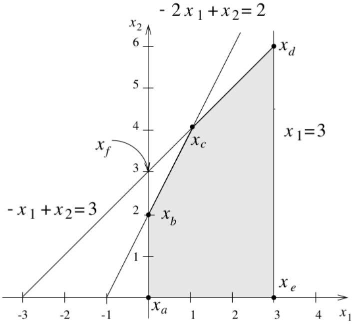
$Ax = b$ where $x \geq 0$

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_3, x_4, x_5\}$ with

$$B = (u_3\ u_4\ u_5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

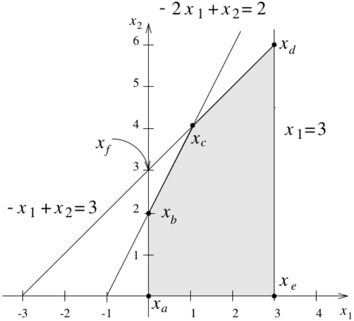
What is x_B satisfying $Bx_B = b$?

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_3, x_4, x_5\}$ with

$$B = (u_3\ u_4\ u_5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is x_B satisfying $Bx_B = b$? $\quad x_B = (x_3, x_4, x_5)^\top = (2, 3, 3)^\top$.

The corresponding basic solution is

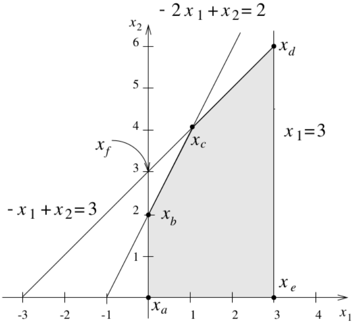
$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 0, 2, 3, 3)^\top = x_a \quad \text{Feasible!}$$

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_2, x_3, x_5\}$ with

$$B = (a_2\ a_3\ a_5) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

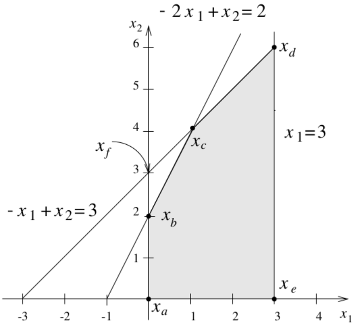
What is x_B satisfying $Bx_B = b$?

$$A = (u_1\, u_2\, u_3\, u_4\, u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_2, x_3, x_5\}$ with

$$B = (a_2\, a_3\, a_5) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

What is x_B satisfying $Bx_B = b$? $x_B = (x_2, x_3, x_5)^\top = (3, -1, 3)^\top$.

The corresponding basic solution is

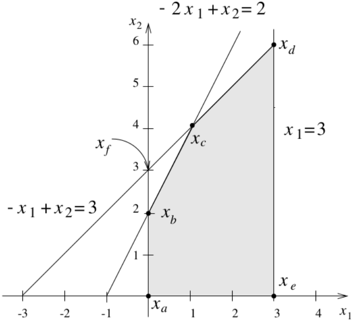
$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (0, 3, -1, 0, 3)^\top = x_f$$ Not feasible!

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_1, x_2, x_3\}$ with

$$B = (u_1\ u_2\ u_3) = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

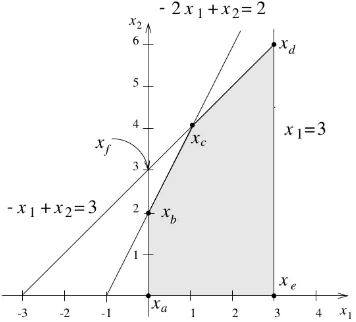
What is x_B satisfying $Bx_B = b$?

$$A = (u_1\ u_2\ u_3\ u_4\ u_5) = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4, x_5)^\top$$

$$Ax = b \text{ where } x \geq 0$$

$$b = (2, 3, 3)^\top$$



Consider a basis $\{x_1, x_2, x_3\}$ with

$$B = (u_1\ u_2\ u_3) = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

What is x_B satisfying $Bx_B = b$? $x_B = (x_1, x_2, x_3)^\top = (3, 6, 2)^\top$.

The corresponding basic solution is

$$x = (x_1, x_2, x_3, x_4, x_5)^\top = (3, 6, 2, 0, 0)^\top = x_d \quad \text{Feasible!}$$

Existence of Basic Feasible Solutions

Theorem 1 (Fundamental Theorem of LP)

Consider a linear program in standard form.

1. *If a feasible solution exists, then a basic feasible solution exists.*
2. *If an optimal feasible solution exists, then an optimal basic feasible solution exists.*

Existence of Basic Feasible Solutions

Theorem 1 (Fundamental Theorem of LP)

Consider a linear program in standard form.

1. *If a feasible solution exists, then a basic feasible solution exists.*
2. *If an optimal feasible solution exists, then an optimal basic feasible solution exists.*

Note that the theorem reduces solving a linear programming problem to searching for basic feasible solutions.

There are finitely many of them, which implies decidability.

Existence of Basic Feasible Solutions

Theorem 1 (Fundamental Theorem of LP)

Consider a linear program in standard form.

1. *If a feasible solution exists, then a basic feasible solution exists.*
2. *If an optimal feasible solution exists, then an optimal basic feasible solution exists.*

Note that the theorem reduces solving a linear programming problem to searching for basic feasible solutions.

There are finitely many of them, which implies decidability.

However, the enumeration of all basic feasible solutions would be impractical; the number of basic feasible solutions is potentially

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

For $n = 100$ and $m = 10$, we get $535,983,370,403,809,682,970$.

Extreme Points

Note that the set Θ of points x satisfying $Ax = b, x \geq 0$ is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

Extreme Points

Note that the set Θ of points x satisfying $Ax = b, x \geq 0$ is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point $x \in \Theta$ is an *extreme point* of Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Extreme Points

Note that the set Θ of points x satisfying $Ax = b, x \geq 0$ is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point $x \in \Theta$ is an *extreme point* of Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Theorem 2

Let Θ be the convex set consisting of all feasible solutions that is, all $x \in \mathbb{R}^n$ satisfying:

$$Ax = b, \quad x \geq 0,$$

where $A \in \mathbb{R}^{m \times n}, m < n$, $rank(A) = m$.
Then, x is an extreme point of Θ if and only if x is a basic feasible solution to $Ax = b, x \geq 0$.

Extreme Points

Note that the set Θ of points x satisfying $Ax = b, x \geq 0$ is *convex polyhedron*.

By definition, a convex hull of a finite set of points.

A point $x \in \Theta$ is an *extreme point* of Θ if there are no two points x' and x'' in Θ such that $x = \alpha x' + (1 - \alpha)x''$ for some $\alpha \in (0, 1)$.

Theorem 2

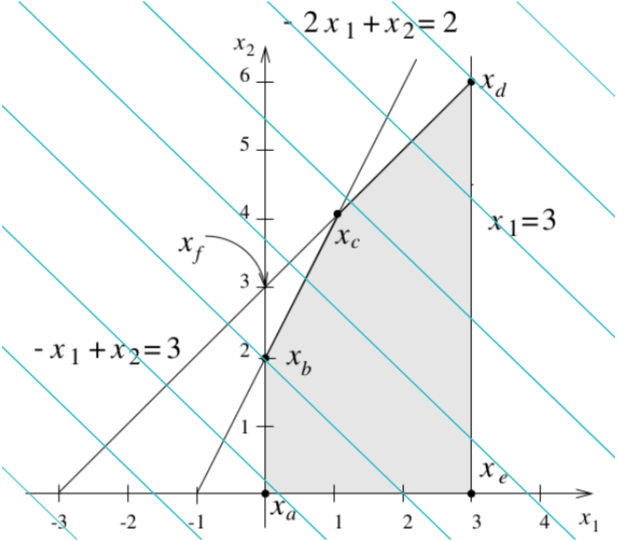
Let Θ be the convex set consisting of all feasible solutions that is, all $x \in \mathbb{R}^n$ satisfying:

$$Ax = b, \quad x \geq 0,$$

where $A \in \mathbb{R}^{m \times n}$, $m < n$, $rank(A) = m$.
Then, x is an extreme point of Θ if and only if x is a basic feasible solution to $Ax = b, x \geq 0$.

Thus, as a corollary, we obtain that to find an optimal solution to the linear optimization problem, we need to consider only extreme points of the feasibility region.

Optimal Solutions



Here, the blue lines are contours of $-x_1 - x_2$. The minimizer is x_d.

Degenerate Basic Solutions

A basic solution $x = [x_B, x_N] \in \mathbb{R}^n$ is *degenerate* if at least one component of x_B is 0.

Degenerate Basic Solutions

A basic solution $x = [x_B, x_N] \in \mathbb{R}^n$ is *degenerate* if at least one component of x_B is 0.

Two different bases can correspond to the same point. To see this, consider the constraints defined by

$$Ax = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 13 \\ 12 \end{pmatrix} = b.$$

Degenerate Basic Solutions

A basic solution $x = [x_B, x_N] \in \mathbb{R}^n$ is *degenerate* if at least one component of x_B is 0.

Two different bases can correspond to the same point. To see this, consider the constraints defined by

$$Ax = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 13 \\ 12 \end{pmatrix} = b.$$

There are two bases

$\{x_1, x_2, x_3\}$ giving

$$B = \begin{pmatrix} 2 & 1 & 0 \\ 3 & 0 & 1 \\ 4 & 0 & 0 \end{pmatrix}$$

$\{x_1, x_3, x_4\}$ giving

$$B' = \begin{pmatrix} 2 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}$$

Each gives the same *degenerate* basic solution $x = (3, 0, 4, 0)^\top$.

Simplex Algorithm

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.
- Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.
- Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.
- Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- If there is no better neighbor, the algorithm stops.

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.
- Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- If there is no better neighbor, the algorithm stops.
- (It may happen that the polyhedron is unbounded if the algorithm finds out that the objective may be infinitely improved.)

Intuition

The algorithm proceeds as follows:

- Start in a vertex of the polyhedron defined by the constraints.
- Move to each of the neighboring vertices and check whether it is better from the point of view of the objective.
- If yes, move to such a neighbor (there may be more than one better than the current one; choose one of them).
- If there is no better neighbor, the algorithm stops.
- (It may happen that the polyhedron is unbounded if the algorithm finds out that the objective may be infinitely improved.)

Now, how do you move from one vertex to another one algebraically?

First, we consider LP problems where each basic solution is non-degenerate.

Later we drop this assumption.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \dots u_m\, u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B \ N) = (u_1 \dots u_m \, u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B \ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \dots, m$.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B \ N) = (u_1 \dots u_m \, u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B \ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \dots, m$.

Now as B is a basis, we have that for each $i \in \{m+1, \dots, n\}$ there are coefficients $y_1, \dots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$.

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \dots u_m\, u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \dots, m$.

Now as B is a basis, we have that for each $i \in \{m+1, \dots, n\}$ there are coefficients $y_1, \dots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots x_m u_m \\ &= x_1 u_1 + \cdots x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Changing Basis (Non-Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \dots u_m\ u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots x_m u_m = b$$

For a non-degenerate case, we have $x_j > 0$ for all $j = 1, \dots, m$.

Now as B is a basis, we have that for each $i \in \{m+1, \dots, n\}$ there are coefficients $y_1, \dots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots x_m u_m \\ &= x_1 u_1 + \cdots x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Now consider maximum $\alpha > 0$ such that $x_j - \alpha y_j \geq 0$ for all j.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\} > 0$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.
The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such j can be computed using:

$$j = \text{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such j can be computed using:

$$j = \text{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j\to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j\to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\} > 0$$

There would be a *unique* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.

The uniqueness follows from non-degeneracy because otherwise, we would move to a basis giving a degenerate solution.

Note that such j can be computed using:

$$j = \text{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

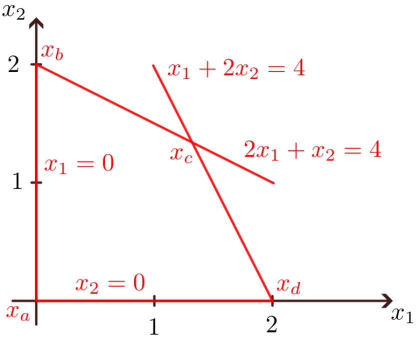
Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.
We say that we *pivot about (j, i)*.

Algorithm 1 Simplex - **Non-degenerate**

Choose a starting basis $B = (u_1 \dots u_m)$ (here $A = (B \ N)$)
repeat
    Compute the basic solution x for the basis B
    for $i \in \{m+1, \dots, n\}$ **do**
        Solve $B(y_1, \dots, y_m)^\top = u_i$
        if $y_k \leq 0$ for all $k \in \{1, \dots, m\}$ **then**
            Stop, unbounded problem.
        end if
        Select $j = \text{argmin}\{x_k / y_k \mid y_k > 0 \land k = 1, \dots, m\}$
        Compute $x_{j \to i}$
    end for
    if $c^\top(x_{j \to i} - x) \geq 0$ for all $i \in \{m+1, \dots, n\}$ **then**
        Stop, we have an optimal solution.
    end if
    Select $i \in \{m+1, \dots, n\}$ such that $c^\top(x_{j \to i} - x) < 0$
    $B \leftarrow B_{j \to i}$
until convergence

$$A = (u_1\ u_2\ u_3\ u_4)$$
$$= \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$
$$b = (4, 4)^\top$$
$$c = (-1, -1, 0, 0)^\top$$



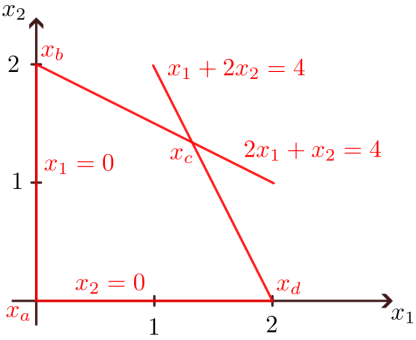
minimize $c^\top x$ subject to $Ax = b$ where $x \geq 0$

$$A = (u_1\; u_2\; u_3\; u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (4, 4)^\top$$

$$c = (-1, -1, 0, 0)^\top$$



minimize $c^\top x$ subject to $Ax = b$ where $x \geq 0$

Consider a basis

$$B = (a_3\; a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The basic solution is $x = (x_1, x_2, x_3, x_4)^\top = (0, 0, 4, 4)^\top$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$, pivot about $(4, 1)$ and $\alpha = x_4/y_4 = 2$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$, pivot about $(4, 1)$ and $\alpha = x_4/y_4 = 2$.

$$x_{4\to1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$, pivot about $(4, 1)$ and $\alpha = x_4/y_4 = 2$.

$$x_{4\to 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis $\{x_1, x_3\}$ and the basic solution $(2, 0, 2, 0)$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$, pivot about $(4, 1)$ and $\alpha = x_4/y_4 = 2$.

$$x_{4\to 1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis $\{x_1, x_3\}$ and the basic solution $(2, 0, 2, 0)$.

Similarly, we may also put x_2 into the basis instead of x_3 and obtain the basis $\{x_2, x_4\}$ and the basic solution $(0, 2, 0, 2)$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Start with the basis $\{x_3, x_4\}$ giving $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 0, 4, 4)$.

Consider x_1 as a candidate to the basis, i.e., consider the first column u_1 of A expressed in the basis B:

$$u_1 = (1, 2)^\top = B\ (1, 2)^\top \text{ thus } y = (y_3, y_4) = (1, 2)$$

Now $x_4/y_4 = 4/2 < 4/1 = x_3/y_3$, pivot about $(4, 1)$ and $\alpha = x_4/y_4 = 2$.

$$x_{4\to1} = (\alpha, 0, (x_3 - \alpha y_3), (x_4 - \alpha y_4)) = (2, 0, 2, 0)$$

As a result we get the basis $\{x_1, x_3\}$ and the basic solution $(2, 0, 2, 0)$.

Similarly, we may also put x_2 into the basis instead of x_3 and obtain the basis $\{x_2, x_4\}$ and the basic solution $(0, 2, 0, 2)$.

We have $c^\top (x_{4\to1} - x) = -2 < 0$
So let us move to the basis $\{x_1, x_3\}$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis $\{x_1, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$.

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis $\{x_1, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$.

Consider x_2 as a candidate for the basis, i.e., consider the second column u_2 of A expressed in the basis B:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis $\{x_1, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$.

Consider x_2 as a candidate for the basis, i.e., consider the second column u_2 of A expressed in the basis B:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$, pivot about $(3, 2)$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis $\{x_1, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$.

Consider x_2 as a candidate for the basis, i.e., consider the second column u_2 of A expressed in the basis B:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$, pivot about $(3, 2)$

$$x_{3\to 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

Non-Degenerate Example

$$c = (-1, -1, 0, 0) \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Consider the basis $\{x_1, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (2, 0, 2, 0)$.

Consider x_2 as a candidate for the basis, i.e., consider the second column u_2 of A expressed in the basis B:

$$u_2 = (2, 1)^\top = B\ (1/2, 3/2)^\top \text{ thus } y = (y_1, y_3) = (1/2, 3/2)$$

Now $\alpha = x_3/y_3 = 4/3 < 2/(1/2) = 4 = x_1/y_1$, pivot about $(3, 2)$

$$x_{3\to 2} = ((x_1 - \alpha y_1), \alpha, (x_3 - \alpha y_3), 0) = (4/3, 4/3, 0, 0)$$

$$c^\top (x_{3\to 2} - x) = c(-2/3, 4/3)^\top = -2/3 < 0$$

We have reached a minimizer. All changes would lead to a higher objective value.

We may exchange x_1 with x_4, but this would give us the initial basis with a higher objective value.

Non-Degenerate Case Convergence

Theorem 3

Suppose that the simplex method is applied to a linear program and that every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.

Non-Degenerate Case Convergence

Theorem 3

Suppose that the simplex method is applied to a linear program and that every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.

However, what happens if we meet a degenerate solution?

Non-Degenerate Case Convergence

Theorem 3

Suppose that the simplex method is applied to a linear program and that every basic variable is strictly positive at every iteration. Then, in a finite number of iterations, the method either terminates at an optimal basic feasible solution or determines that the problem is unbounded.

However, what happens if we meet a degenerate solution?

So, let us drop the non-degeneracy assumption.

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \dots u_m\, u_{m+1} \dots u_n)$ where $B = (u_1 \dots u_m)$ and $N = (u_{m+1} \dots u_n)$.

Note that each u_i is a column vector of dimension m.

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \ldots, m\}$, and *may have* $x_i = 0$ *for some* $j \in \{1, \ldots, m\}$.

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \ldots, m\}$, and *may have* $x_i = 0$ *for some* $j \in \{1, \ldots, m\}$.

Now as B is a basis, we have that for each $i \in \{m+1, \ldots, n\}$ there are coefficients $y_1, \ldots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$.

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \ldots, m\}$, and *may have $x_i = 0$ for some $j \in \{1, \ldots, m\}$*.

Now as B is a basis, we have that for each $i \in \{m+1, \ldots, n\}$ there are coefficients $y_1, \ldots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots + x_m u_m \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Changing Basis (Degenerate Case)

Consider a basis B and write $A = (B\ N) = (u_1 \ldots u_m\, u_{m+1} \ldots u_n)$ where $B = (u_1 \ldots u_m)$ and $N = (u_{m+1} \ldots u_n)$.

Note that each u_i is a column vector of dimension m.

Consider a basic feasible solution $x = [x_B\ x_N]$ where $x_N = 0$. Then

$$x_1 u_1 + \cdots + x_m u_m = b$$

For a degenerate case, we have $x_j \geq 0$ for all $j \in \{1, \ldots, m\}$, and *may have* $x_i = 0$ *for some* $j \in \{1, \ldots, m\}$.

Now as B is a basis, we have that for each $i \in \{m+1, \ldots, n\}$ there are coefficients $y_1, \ldots, y_m$ such that $y_1 u_1 + \cdots + y_m u_m = u_i$. Then

$$\begin{aligned} b &= x_1 u_1 + \cdots + x_m u_m \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha u_i + \alpha u_i \\ &= x_1 u_1 + \cdots + x_m u_m - \alpha(y_1 u_1 + \cdots + y_m u_m) + \alpha u_i \\ &= (x_1 - \alpha y_1) u_1 + \cdots + (x_m - \alpha y_m) u_m + \alpha u_i \end{aligned}$$

Now consider maximum $\alpha \geq 0$ such that $x_j - \alpha y_j \geq 0$ for all j.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.
j DOES NOT have to be unique in a degenerate case.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.

j DOES NOT have to be unique in a degenerate case.

Note that such j can be computed using:

$$j \in \text{argmin}\{x_k/y_k \mid y_k > 0 \land k = 1, \ldots, m\}$$

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.
j DOES NOT have to be unique in a degenerate case.

Note that such j can be computed using:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j \to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j \to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.
Note that if $\alpha = 0$, the solution does not change. The basis, however, changes.

$$b = (x_1 - \alpha y_1)u_1 + \cdots + (x_m - \alpha y_m)u_m + \alpha u_i$$

If all $y_j \leq 0$, the problem is unbounded because one component grows indefinitely and others do not decrease with $\alpha \to \infty$.

Otherwise, we put

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Otherwise, there *exists* $j \in \{1, \ldots, m\}$ such that $x_j - \alpha y_j = 0$.

j DOES NOT have to be unique in a degenerate case.

Note that such j can be computed using:

$$j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \ldots, m\}$$

Obtain a basis $B_{j\to i} = B \smallsetminus \{j\} \cup \{i\}$ and a basic feasible solution

$$x_{j\to i} = (x'_1, \ldots, x'_{j-1}, 0, x'_{j+1}, \ldots, x'_m, 0, \ldots, 0, \alpha, 0, \ldots, 0)^\top$$

Here $x'_k = x_k - \alpha y_k$ for each $k \in \{1, \ldots, j-1, j+1, \ldots, m\}$.

Note that if $\alpha = 0$, the solution does not change. The basis, however, changes.

We say that we *pivot about* (j, i).

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 1$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 1$

$$x_{2\to4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^\top = (0, 0, 1, 1)^\top$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 1$

$$x_{2\to 4} = (0, (x_2 - \alpha y_2), (x_3 - \alpha y_3), \alpha)^\top = (0, 0, 1, 1)^\top$$

Note that $c^\top x_{2\to 4} = 0$.

Thus **no effect on the objective value!**

Degenerate Example

$$c = (-1,0,0,0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0,1,0,0)^\top$ with $c^\top x = 0$.

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about $(3, 1)$, that is x_3 exchanges with x_1 and $\alpha = x_3/y_3 = 0$.

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about $(3, 1)$, that is x_3 exchanges with x_1 and $\alpha = x_3/y_3 = 0$.

$$x_{3\to 1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\; u_2\; u_3\; u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about $(3, 1)$, that is x_3 exchanges with x_1 and $\alpha = x_3/y_3 = 0$.

$$x_{3\to1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

No change in the basic solution, and thus $c^\top x_{3\to1} = c^\top x = 0$.

Thus **no effect on the objective value either!**

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4)^\top = (0, 1, 0, 0)^\top$ with $c^\top x = 0$.

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Pivot about $(3, 1)$, that is x_3 exchanges with x_1 and $\alpha = x_3/y_3 = 0$.

$$x_{3\to1} = (\alpha, (x_2 - \alpha y_2), (x_3 - \alpha y_3), 0)^\top = (0, 1, 0, 0)^\top$$

No change in the basic solution, and thus $c^\top x_{3\to1} = c^\top x = 0$.

Thus **no effect on the objective value either!**

Which variable should go to the basis?!

Reduced Cost

Given a basis B, we denote by c_B the vector of components of c that correspond to the variables of B.

Reduced Cost

Given a basis B, we denote by c_B the vector of components of c that correspond to the variables of B.

One can prove that for every $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j \to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ where $By = u_i$.

Reduced Cost

Given a basis B, we denote by c_B the vector of components of c that correspond to the variables of B.

One can prove that for every $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j \to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ where $By = u_i$.

For non-degenerate case, we have $\alpha > 0$ and thus

$$c^\top x_{j \to i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have $\alpha = 0$ and $c_i - c_B y < 0$.

Reduced Cost

Given a basis B, we denote by c_B the vector of components of c that correspond to the variables of B.

One can prove that for every $i \in \{m+1, \ldots, n\}$ we have

$$c^\top x_{j\to i} - c^\top x = (c_i - c_B^\top y)\alpha$$

Here $y = (y_1, \ldots, y_m)^\top$ where $By = u_i$.

For non-degenerate case, we have $\alpha > 0$ and thus

$$c^\top x_{j\to i} < c^\top x \quad \text{iff} \quad c_i - c_B^\top y < 0$$

For the degenerate case, we may have $\alpha = 0$ and $c_i - c_B y < 0$.

Define the *reduced cost* by

$$r_i = c_i - c_B^\top y$$

Intuitively, c_i is the cost of x_i in the new basis and $c_B^\top y$ in the old one.

Derivation of Reduced Cost

$$
\begin{aligned}
c^\top x_{j\to i} &= c^\top(x'_1,\ldots,x'_{j-1},0,x'_{j+1},\ldots,x'_m,0,\ldots,0,\alpha,0,\ldots,0)^\top \\
&= c^\top(x'_1,\ldots,x'_{j-1},x'_j,x'_{j+1},\ldots,x'_m,0,\ldots,0,\alpha,0,\ldots,0)^\top \\
&= c_1x'_1+\cdots+c_mx'_m+c_i\alpha \\
&= c_1(x_1-\alpha y_1)+\cdots c_m(x_m-\alpha y_m)+c_i\alpha \\
&= (c_1x_1+\cdots+c_mx_m)-(c_1y_1+\cdots+c_my_m-c_i)\alpha \\
&= c^\top x-(-c_i+c_By)\alpha
\end{aligned}
$$

Here we use the fact that $x'_k = x_k - \alpha y_k$ for each $k \in \{1,\ldots,j-1,j+1,\ldots,m\}$ and that $x_j - \alpha y_j = 0$.

Then clearly

$$c^\top x_{j\to i} - c^\top x = (c_i - c_By)\alpha$$

$$\alpha = \min\{x_k/y_k \mid y_k > 0 \wedge k = 1,\ldots,m\}$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

The reduced cost is

$$r_1 = c_1 - (c_2 y_2 + c_3 y_3) = -1 - (0 \cdot (-1) + 0 \cdot 2) = -1 < 0$$

Degenerate Example

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Start with the basis $\{x_2, x_3\}$ giving $B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $cx = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(1, -1)^\top \text{ thus } y = (y_2, y_3) = (1, -1)$$

The reduced cost is:

$$r_4 = c_4 - (c_2 y_2 + c_3 y_3) = 0 - (0 \cdot 1 + 0 \cdot (-1)) = 0$$

Consider x_1 as a candidate for the basis:

$$u_1 = (1, -1)^\top = B(-1, 2)^\top \text{ thus } y = (y_2, y_3) = (-1, 2)$$

The reduced cost is

$$r_1 = c_1 - (c_2 y_2 + c_3 y_3) = -1 - (0 \cdot (-1) + 0 \cdot 2) = -1 < 0$$

So we should put x_1 into the basis (the reduced cost gets smaller).

Algorithm 2 Simplex

Choose a starting basis $B = (u_1 \dots u_m)$ (here $A = (B \; N)$)
repeat
    Compute the basic solution x for the basis B
    for $i \in \{m+1, \dots, n\}$ **do**
        Solve $B\,(y_1, \dots, y_m)^\top = u_i$
        if $y_k \le 0$ for all $k \in \{1, \dots, m\}$ **then**
            Stop, unbounded problem.
        end if
        Select $j \in \operatorname{argmin}\{x_k/y_k \mid y_k > 0 \wedge k = 1, \dots, m\}$
        Compute $r_i = c_i - c_B^\top y$ where $y = (y_1, \dots, y_m)^\top$
    end for
    if $r_i \ge 0$ for all $i \in \{m+1, \dots, n\}$ **then**
        Stop, we have an optimal solution.
    end if
    Select $i \in \{m+1, \dots, n\}$ such that $r_i < 0$
    $B \leftarrow B_{j \to i}$
until convergence

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\; u_2\; u_3\; u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 2$

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Does this always work?

Degenerate Example (Cont.)

$$c = (-1, 0, 0, 0)^\top \quad A = (u_1\, u_2\, u_3\, u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After following the reduced cost from the basis $\{x_2, x_3\}$, we end up in the basis $\{x_1, x_2\}$ giving $B = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ and the basic solution $x = (x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ with $c^\top x = 0$.

Consider x_4 as a candidate for the basis:

$$u_4 = (0, 1)^\top = B\,(-1/2, 1/2)^\top \text{ thus } y = (y_1, y_2) = (-1/2, 1/2)$$

Pivot about $(2, 4)$, that is x_2 exchanges with x_4 and $\alpha = x_2/y_2 = 2$

$$x_{2\to 4} = ((x_1 - \alpha y_1), (x_2 - \alpha y_2), 0, \alpha) = (1, 0, 0, 2)$$

This is the minimizer!

Does this always work? Unfortunately, NO!

Degenerate Case - Looping

Consider the following linear program:

$$\begin{array}{ll} \text{minimize} & z = -\frac{3}{4}x_1 + 150x_2 - \frac{1}{50}x_3 + 6x_4 \\ \text{subject to} & \frac{1}{4}x_1 - 60x_2 - \frac{1}{25}x_3 + 9x_4 + x_5 = 0 \\ & \frac{1}{2}x_1 - 90x_2 - \frac{1}{50}x_3 + 3x_4 + x_6 = 0 \\ & x_3 + x_7 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0 \end{array}$$

Executing the simplex method on this program starting with the basis $\{x_5, x_6, x_7\}$ and always choosing i minimizing the reduced cost at line 15, eventually ends up back in the basis $\{x_5, x_6, x_7\}$.

In other words, even though the reduced cost is always negative, the overall effect on the objective is 0.

Convergence of Simplex Method

A solution is to use Bland's rule:

- Select the smallest index j at line 9.
- Select the smallest index i at line 15.

Theorem 4

If the simplex method is implemented using Bland's rule to select the entering and leaving variables, then the simplex method is guaranteed to terminate.

Simplex Convergence Summary

In a non-degenerate case:

- There is always a unique j to be selected at line 9.
- The objective of the basic solution decreases with each step.

Thus, we have a deterministic algorithm that always terminates in a non-degenerate case.

Simplex Convergence Summary

In a non-degenerate case:

- There is always a unique j to be selected at line 9.
- The objective of the basic solution decreases with each step.

Thus, we have a deterministic algorithm that always terminates in a non-degenerate case.

In a degenerate case:

- We may have several j from which to select at line 9.
- Even though the reduced cost is negative, the basic solution may remain the same.

The simplex algorithm may cycle!

Using Bland's rule, the simplex method always converges to a minimizer or detects an unbounded LP.

Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

How do we obtain such a solution? Given a standard form LP

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Two-Phase Simplex Algorithm

A Simplex algorithm is initialized with a basic feasible solution.

How do we obtain such a solution? Given a standard form LP

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We construct an artificial LP problem.

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + \cdots + y_m \\ \text{subject to} \quad & (A \ I_m) \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0 \end{aligned}$$

Here $y = (y_1, \ldots, y_m)^\top$ is a vector of artificial variables, I_m is the identity matrix of dimensions $m \times m$.

Two-Phase Simplex Algorithm

Solve the artificial LP problem:

$$\begin{aligned} \text{minimize} \quad & y_1 + y_2 + \cdots + y_m \\ \text{subject to} \quad & [A \ I_m] \begin{pmatrix} x \\ y \end{pmatrix} = b \\ & \begin{pmatrix} x \\ y \end{pmatrix} \geq 0 \end{aligned}$$

Proposition 1

The original LP problem has a basic feasible solution iff the associated artificial LP problem has an optimal feasible solution with the objective function 0.

If we solve the artificial problem with $y = 0$, we obtain x such that $Ax = b, x \geq 0$ is a basic feasible solution for the original problem.

If there is no such a solution to the artificial problem, there is no basic feasible solution, and hence no feasible solution, to the original problem.

Linear Programming
Properties

LP Complexity

Iterations of the simplex algorithm can be implemented to compute the first step using $\mathcal{O}(m^2 n)$ arithmetic operations and each next step $\mathcal{O}(mn)$.

LP Complexity

Iterations of the simplex algorithm can be implemented to compute the first step using $\mathcal{O}(m^2 n)$ arithmetic operations and each next step $\mathcal{O}(mn)$.

There are as many as $\binom{n}{m}$ basic solutions (many of them likely infeasible). How large are these numbers?

m	$\binom{2m}{m}$
1	2
5	252
10	184756
20	1×10^{11}
50	1×10^{29}
100	9×10^{58}
200	1×10^{119}
300	1×10^{179}
400	2×10^{239}
500	3×10^{299}

The number of iterations may be proportional to $\binom{n}{m}$ that is EXPTIME.

Linear Programming Complexity

Complexity of the simplex algorithm:

- In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.
  For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.

Linear Programming Complexity

Complexity of the simplex algorithm:

- In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.
  For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.
- There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)

Linear Programming Complexity

Complexity of the simplex algorithm:

- In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.
  For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.
- There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)

Linear Programming Complexity

Complexity of the simplex algorithm:

- In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.

  For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.

- There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)
  - Then, the expected computation time for the resulting instances of LP is polynomial.

  For details, see "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time" by Daniel A. Spielman and Shang-Hua Teng in JACM 2004.

Linear Programming Complexity

Complexity of the simplex algorithm:

- In the worst case, the time complexity of the simplex algorithm is exponential. This holds for any deterministic pivoting rule.

  For details, see "How good is the simplex algorithm?" by Klee, Victor, and Minty, George J. Inequalities 1972.
- There is a theory that shows that examples with exponential complexity are rare. More precisely (but still very imprecisely)
  - Consider small random perturbations of the coefficients in the LP (use Gaussian noise with a small variance)
  - Then, the expected computation time for the resulting instances of LP is polynomial.

  For details, see "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time" by Daniel A. Spielman and Shang-Hua Teng in JACM 2004.

Is there a deterministic polynomial time algorithm for solving LP?

Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

Theorem 5 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)

There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.

The algorithm uses so-called ellipsoid method.

Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

Theorem 5 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)

There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.

The algorithm uses so-called ellipsoid method.

In practice, the Khachiyan's is not used.

Linear Programming Complexity

We assume that all coefficients are encoded in binary (more precisely, as fractions of two integers encoded in binary).

Theorem 5 (Khachiyan, Doklady Akademii Nauk SSSR, 1979)

There is an algorithm that, for any linear program, computes an optimal solution in polynomial time.

The algorithm uses so-called ellipsoid method.

In practice, the Khachiyan's is not used.

There is also a polynomial time algorithm (by Karmarkar) that has lower complexity upper bounds than the Khachiyan's and sometimes works even better than the simplex.

Linear Programming in Practice

Heavily used tools for solving practical problems.

Several advanced linear programming solvers (usually parts of larger optimization packages) implement various heuristics for solving large-scale problems, such as sensitivity analysis.

See an overview of tools here:

http://en.wikipedia.org/wiki/Linear_programming#Solvers_and_scripting_.28programming.29_languages

For example, the well-known Gurobi solver uses the simplex algorithm to solve LP problems.

Linear Programming - Tableaus

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

Tableau

Consider a linear program in the standard form:

$$\begin{array}{rl} \text{minimize} & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

The algorithm is relatively straightforward but, in its original form, not so suitable for computations by hand.

Tableau

Consider a linear program in the standard form:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

We have considered the simplex algorithm, which searches for the minimum by moving around the vertices of the feasible region.

The algorithm is relatively straightforward but, in its original form, not so suitable for computations by hand.

Tableaus provide all information about the current state of the simplex algorithm and can be used to streamline the process.

Keep in mind that we are not developing a new algorithm. Tableau just provides another view of the same simplex algorithm as presented before.

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\ N)$ where B consists of basic columns and N of the non-basic ones.

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\ N)$ where B consists of basic columns and N of the non-basic ones.

Consider the following matrix (the *initial tableau*):

$$\begin{pmatrix} A & b \\ c^\top & 0 \end{pmatrix} = \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

Consider LP with a matrix A and vectors b, c. Assume $A = (B\ N)$ where B consists of basic columns and N of the non-basic ones.

Consider the following matrix (the *initial tableau*):

$$\begin{pmatrix} A & b \\ c^\top & 0 \end{pmatrix} = \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Apply elementary row operations so that the matrix B is turned into I_m (preserving the last row for now). That is, multiply with

$$\begin{pmatrix} B^{-1} & 0 \\ 0 & 1 \end{pmatrix}$$

The result is

$$\begin{pmatrix} B^{-1} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

We apply row operations to the last row to eliminate the $c_B^\top$. This corresponds to multiplying the matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix}$$

Tableau (Matrix Form)

We have

$$\begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix}$$

We apply row operations to the last row to eliminate the $c_B^\top$. This corresponds to multiplying the matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix}$$

We obtain

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix} \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} I_m & B^{-1}N & B^{-1}b \\ 0 & c_N^\top - c_B^\top B^{-1}N & -c_B^\top B^{-1}b \end{pmatrix}$$

This is the *canonical form tableau for the basis B*.

Tableau (Components)

Let $A = (u_1 \ \ldots, u_n)$, the basis $\{x_1, \ldots, x_m\}$, $B = (u_1 \ \ldots, u_m)$.

Assume $u_k = (u_{1k}, \ldots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

Let $A = (u_1 \dots, u_n)$, the basis $\{x_1, \dots, x_m\}$, $B = (u_1 \dots, u_m)$.

Assume $u_k = (u_{1k}, \dots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Now transform all columns of the upper part of the matrix (except the last row) to the basis B:

$$u_k = B(y_{1k}, \dots, y_{mk})^\top \text{ for } k = 1, \dots, n \text{ and } b' = B^{-1}b$$

Tableau (Components)

Let $A = (u_1 \ldots, u_n)$, the basis $\{x_1, \ldots, x_m\}$, $B = (u_1 \ldots, u_m)$.

Assume $u_k = (u_{1k}, \ldots, u_{nk})$. Then the initial tableau is

$$\begin{pmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & \cdots & u_{1m} & u_{1(m+1)} & \cdots & u_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{m1} & \cdots & u_{mm} & u_{m(m+1)} & \cdots & u_{mn} & b_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Now transform all columns of the upper part of the matrix (except the last row) to the basis B:

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

and obtain $u_k = y_{1k}u_1 + \cdots + y_{mk}u_m$ for $k = m+1, \ldots, n$ and thus

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Tableau (Components)

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ c_1 & \cdots & c_m & c_{m+1} & \cdots & c_n & 0 \end{pmatrix}$$

Use row operations to eliminate $c_1, \ldots, c_m$. This is equivalent to multiplying the above matrix with

$$\begin{pmatrix} I_m & 0 \\ -c_B^\top & 1 \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \\ -c_1 & \cdots & -c_m & 1 \end{pmatrix}$$

from the left. We obtain ...

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Here, $(b'_1, \ldots, b'_m)^\top = B^{-1}b$ is the vector b transformed to the basis B, and for $k = m+1, \ldots, n$ we have

$$c'_k = c_k - (y_{1k}c_1 + \cdots + y_{mk}c_m)$$

the reduced cost for the k-th column (non-basic).

Tableau (Components)

... the canonical form for the basis $\{x_1, \ldots, x_m\}$:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

Here, $(b'_1, \ldots, b'_m)^\top = B^{-1}b$ is the vector b transformed to the basis B, and for $k = m+1, \ldots, n$ we have

$$c'_k = c_k - (y_{1k}c_1 + \cdots + y_{mk}c_m)$$

the reduced cost for the k-th column (non-basic). Also, note that the basic solution is $x = (b'_1, \ldots, b'_m, 0, \ldots, 0)$, and hence

$$-z = (-c_1)b'_1 + \cdots + (-c_m)b'_m$$

is the negative of the value of the objective for the basic solution corresponding to the basis $\{x_1, \ldots, x_m\}$.

Recall that, by definition, the basic solution x satisfies $x_{m+1} = \cdots = x_n = 0$.

Tableau Simplex

Assume that for a basis B we have obtained the canonical tableau:

$$\begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \\ 0 & \cdots & 0 & c'_{m+1} & \cdots & c'_n & -z \end{pmatrix}$$

The simplex algorithm then proceeds as follows:

1. Choose $i \in \{m+1, \ldots, n\}$ such that $c'_i < 0$.
2. Choose $j \in \{1, \ldots, m\}$ minimizing b'_j / y_{ji} over all j satisfying $y_{ji} > 0$.

   Note that $b'_j = x_j$ for the basic solution x w.r.t. B.
3. Move the i-the column into the basis and the j-th column out of the basis.
4. Use elementary row operations to transform the tableau into the canonical form for the new basis.
5. Repeat until $b'_1, \ldots, b'_m \geq 0$,

Example

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$x = (x_1, x_2, x_3, x_4)^\top$

$b = (2, 1)^\top$

$Ax = b$ where $x \geq 0$

$c = (-3, -2, 0, 0)^\top$

Example

Add slack variables x_3, x_4:

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 &\leq 1 \\ x_1, x_2 &\geq 0 \end{aligned}$$

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$A = (u_1\ u_2\ u_3\ u_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$x = (x_1, x_2, x_3, x_4)^\top$$

$$b = (2, 1)^\top$$

$Ax = b$ where $x \geq 0$

$$c = (-3, -2, 0, 0)^\top$$

Tableau for the basis $\{x_3, x_4\}$:

$$\left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

is already in the canonical form.

Note that the last row of the tableau corresponds to writing the objective as $-z + c^\top x = 0$ where z is a new variable and x is the basic solution for $\{x_3, x_4\}$.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b'_1 \\ x_3 & 0 & y_{32} & 1 & y_{34} & b'_2 \\ \hline -z & c'_1 & c'_2 & c'_3 & c'_4 & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b'_1 \\ x_3 & 0 & y_{32} & 1 & y_{34} & b'_2 \\ \hline -z & c'_1 & c'_2 & c'_3 & c'_4 & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b'_1 \\ x_3 & 0 & y_{32} & 1 & y_{34} & b'_2 \\ \hline -z & c'_1 & c'_2 & c'_3 & c'_4 & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.Now x_3 leaves the basis because $y_{12} = 0$ but $y_{32} > 0$.

Start with the basis $\{x_3, x_4\}$ and consider the canonical form:

$$\left[\begin{array}{c|cccc|c} x_3 & y_{31} & y_{32} & 1 & 0 & b_1 \\ x_4 & y_{41} & y_{42} & 0 & 1 & b_2 \\ \hline -z & c_1 & c_2 & c_3 & c_4 & 0 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_3 & 1 & 1 & 1 & 0 & 2 \\ x_4 & 1 & 0 & 0 & 1 & 1 \\ \hline -z & -3 & -2 & 0 & 0 & 0 \end{array}\right]$$

Choose x_1 to enter the basis (x_1 has the reduced cost -3 and x_2 has the reduced costs -2).Now $b_1/y_{31} = 2/1 > 1/1 = b_2/y_{41}$. Thus, remove x_4 from the basis.We move to the basis $\{x_1, x_3\}$ and transform the tableau into the canonical form for this basis:

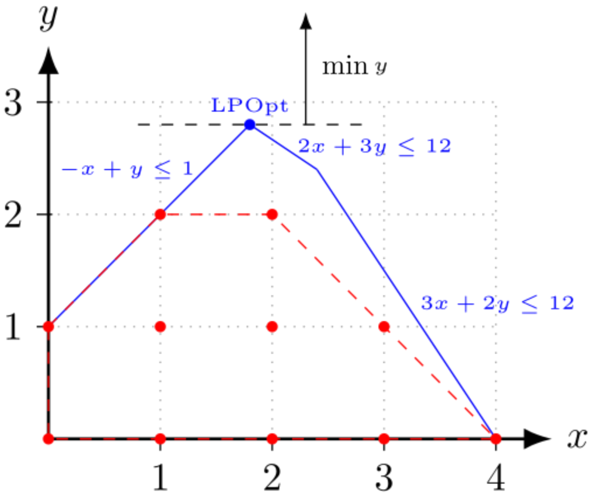
$$\left[\begin{array}{c|cccc|c} x_1 & 1 & y_{12} & 0 & y_{14} & b_1' \\ x_3 & 0 & y_{32} & 1 & y_{34} & b_2' \\ \hline -z & c_1' & c_2' & c_3' & c_4' & 3 \end{array}\right] = \left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_3 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & -2 & 0 & 3 & 3 \end{array}\right]$$

Here, the reduced cost of x_2 is -2, and of x_4 is 3. Thus, x_2 enters the basis.Now x_3 leaves the basis because $y_{12} = 0$ but $y_{32} > 0$.We move to the basis $\{x_1, x_2\}$ and transform the tableau into the canonical form:

$$\left[\begin{array}{c|cccc|c} x_1 & 1 & 0 & 0 & 1 & 1 \\ x_2 & 0 & 1 & 1 & -1 & 1 \\ \hline -z & 0 & 0 & 2 & 1 & 5 \end{array}\right]$$

Integer Linear Programming

Integer Linear Programming



ILP = LP + variables constrained to integer values

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

We also consider a cutting-plane method for integer programming.

Integer Linear Programming

We consider several variants of integer programming:

- 0-1 integer linear programming
- Mixed 0-1 integer linear programming
- Integer linear programming
- Mixed integer linear programming

We consider the basic branch and bound algorithm.

We also consider a cutting-plane method for integer programming.

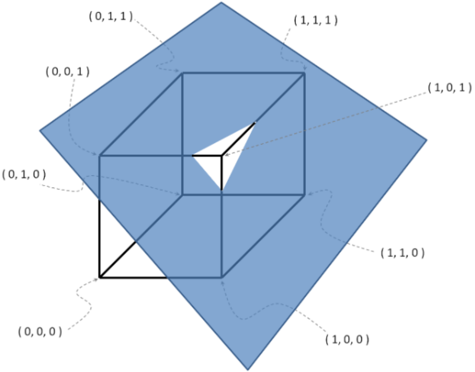
Integer linear programming is a huge subject; we shall only scratch its surface slightly.

0-1 Integer Linear Programming

Let us start with a special case where variables are constrained to values from $\{0, 1\}$.

0-1 integer linear program (0-1 ILP) is

$$
\begin{aligned}
\text{minimize} \quad & c^\top x \\
\text{subject to} \quad & Ax \leq b \\
& x_i \in \{0, 1\}
\end{aligned}
$$



0-1 Integer Linear Programming

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{aligned}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem?

0-1 Integer Linear Programming

Consider the following example:

$$\begin{array}{rl} \text{minimize} & c^\top x \\ \text{subject to} & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{array}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem? It is the 0-1 knapsack problem.

0-1 Integer Linear Programming

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & a^\top x \leq b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \end{aligned}$$

Here $c, a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Do you recognize the problem? It is the 0-1 knapsack problem.

Theorem 6

Finding $x \in \{0, 1\}^n$ satisfying the constraints of a given 0-1 integer linear program is NP-complete.

It is one of Karp's 21 NP-complete problems.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$
\begin{aligned}
\text{minimize} \quad & c^\top x \\
\text{subject to} \quad & Ax = b \\
& x \geq 0 \\
& x_i \in \{0, 1\} \text{ for } x_i \in \mathcal{D}
\end{aligned}
$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0,1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0, 1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

The problem can be solved by searching for possible values 0 and 1 in the binary variables and solving the linear programs with binary variables fixed to concrete values.

0-1 Mixed Integer Linear Programming

0-1 mixed integer linear program (0-1 MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \{0,1\} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of *binary variables*.

The problem is NP-hard; the simplex algorithm cannot be used directly.

The problem can be solved by searching for possible values 0 and 1 in the binary variables and solving the linear programs with binary variables fixed to concrete values.

An exhaustive search through all possible binary assignments would be infeasible for many variables.

Usually, a sequential search that fixes only some of the binary variables and leaves the rest unrestricted to 0 or 1 is used.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0,1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the 0-1 MILP constraints. Initialize with $f^* = \infty$.

Notation

In what follows, *LP relaxation* is the linear program obtained from 0-1 MILP by removing the constraints $x_i \in \{0, 1\}$ for $x_i \in \mathcal{D}$ and adding constraints $x_i \geq 0$ and $x \leq 1$ for all $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the 0-1 MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the 0-1 MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of 0-1 MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original 0-1 MILP to be solved.

Algorithm 3 Branch and Bound (Non-Deterministic)

```
repeat
    Choose $P \in \mathcal{P}$
    if LP relaxation of $P$ is feasible then
        Find a solution $x$ of the LP relaxation of $P$
        if $c^\top x < f^*$ then
            if $x_i \in \{0, 1\}$ for all $x_i \in \mathcal{D}$ then
                $x^* \leftarrow x$
                $f^* \leftarrow c^\top x$
            else
                Choose $x_i \in \mathcal{D}$ such that $x_i \notin \{0, 1\}$
                Generate LP $P_0$ by adding $x_i = 0$ to $P$
                Generate LP $P_1$ by adding $x_i = 1$ to $P$
                Add $P_0$ and $P_1$ to $\mathcal{P}$.
            end if
        end if
    end if
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$
until $\mathcal{P} = \emptyset$
```

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

The solutions to the LP relaxations can be reused. Some methods (dual simplex) exploit that we are just adding a single constraint $x_i = 0$ or $x_i = 1$.

Strategies

There are many possible strategies for choosing the problem to be solved next:

- DFS, BFS, etc.
- heuristics using solutions to the relaxations

There are heuristics for choosing the variable to be bounded:

- Simplest one: Choose x_i which maximizes $\min\{x_i, 1 - x_i\}$
- Look ahead to the relaxations of the possible subdivisions

The solutions to the LP relaxations can be reused. Some methods (dual simplex) exploit that we are just adding a single constraint $x_i = 0$ or $x_i = 1$.

The procedure may be stopped when we find a solution x, which gives a small enough value of the objective.

(Mixed) Integer Programming

Integer linear program (ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

(Mixed) Integer Programming

Integer linear program (ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

Mixed integer linear program (MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of integer variables.

(Mixed) Integer Programming

Integer linear program (ILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

Mixed integer linear program (MILP) is

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \text{ for } x_i \in \mathcal{D} \end{aligned}$$

Here $\mathcal{D} \subseteq \{x_1, \ldots, x_n\}$ is a set of integer variables.

We may use a similar branch and bound approach as for the binary variables. The problem is that now, each integer variable has an infinite domain.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original MILP to be solved.

Notation

In what follows, *LP relaxation* is the linear program obtained from MILP by removing the constraints $x_i \in \mathbb{Z}$ for $x_i \in \mathcal{D}$.

Assume a global variable x^*, keeping the best solution satisfying the MILP constraints. Initialized with the undefined symbol $\perp$.

Assume a global variable f^*, keeping the value of the best solution satisfying the MILP constraints. Initialize with $f^* = \infty$.

Keep a pool of MILP problems $\mathcal{P}$ initialized with $\mathcal{P} = \{P\}$ where P is the original MILP to be solved.

In what follows, we temporarily cease to abuse notation and use $\bar{x}$ to denote the vector of values of the vector of variables x. Then $\bar{x}_i$ will denote the concrete value of the variable x_i.

Algorithm 4 Branch and Bound (Non-Deterministic)

repeat
  Choose $P \in \mathcal{P}$
  if LP relaxation of P is feasible **then**
    Find a solution $\bar{x}$ of the LP relaxation of P
    if $c^\top \bar{x} < f^*$ **then**
      if $\bar{x}_i \in \mathbb{Z}$ for all $x_i \in \mathcal{D}$ **then**
        $x^* \leftarrow \bar{x}$
        $f^* \leftarrow c^\top \bar{x}$
      else
        Choose $x_i \in \mathcal{D}$ such that $\bar{x}_i \notin \mathbb{Z}$
        Generate LP P_- by adding $x_i \leq \lfloor \bar{x}_i \rfloor$ to P
        Generate LP P_+ by adding $x_i \geq \lceil \bar{x}_i \rceil$ to P
        Add P_0 and P_1 to $\mathcal{P}$.
      end if
    end if
  end if
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$
until $\mathcal{P} = \emptyset$

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

The solution to the LP relaxation of P is:

$$x = [0, 1.1818, 4.4091, 0], \quad \text{the objective value is } -15.59$$

Example

Consider the following MILP P:

$$\begin{array}{ll} \text{minimize} & -x_1 - 2x_2 - 3x_3 - 1.5x_4 \\ \text{subject to} & x_1 + x_2 + 2x_3 + 2x_4 \leq 10 \\ & 7x_1 + 8x_2 + 5x_3 + x_4 = 31.5 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{array}$$

and assume $\mathcal{D} = \{x_1, x_2, x_3\}$. That is, $x_1, x_2, x_3 \in \mathbb{Z}$.

The algorithm starts with $\mathcal{P} = \{P\}$ and $x^* = \perp$ and $f^* = \infty$.

The solution to the LP relaxation of P is:

$$x = [0, 1.1818, 4.4091, 0], \quad \text{the objective value is } -15.59$$

Let us choose x_3. So, consider two programs:

- P_- where we add $x_3 \leq 4$ to P
- P_+ where we add $x_3 \geq 5$ to P

Now $\mathcal{P} = \{P_-, P_+\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \qquad \text{the objective value is } -15.25$$

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \qquad \text{the objective value is } -15.25$$

We still have $f^* = \infty$ so we split P_- by constraining x_2:

- P_{--} is obtained from P_- by adding $x_2 \leq 1$
- P_{-+} is obtained from P_- by adding $x_2 \geq 2$

and we continue with $\mathcal{P} = \{P_{--}, P_{-+}\}$.

Consider first P_+.

P_+ is P with the added constraint $x_3 \geq 5$. The LP relaxation of P_+ is infeasible. We get $\mathcal{P} = \{P_-\}$.

P_- is P with the additional constraint $x_3 \leq 4$.

The LP relaxation of P_- solves to

$$\bar{x} = [0, 1.4, 4, 0.3], \qquad \text{the objective value is } -15.25$$

We still have $f^* = \infty$ so we split P_- by constraining x_2:

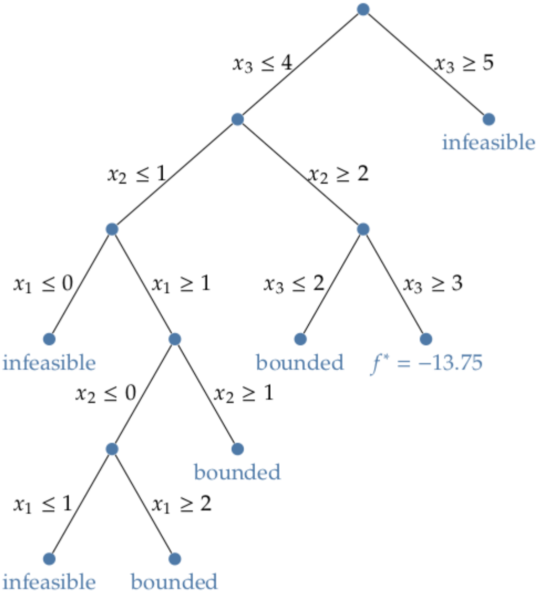
- P_{--} is obtained from P_- by adding $x_2 \leq 1$
- P_{-+} is obtained from P_- by adding $x_2 \geq 2$

and we continue with $\mathcal{P} = \{P_{--}, P_{-+}\}$.

Adding one more constraint $x_3 \geq 3$ to P_{-+} would yield a MILP solution $(0, 2, 3, 0.5)$ to the LP relaxation with the objective value equal to -13.75.

The algorithm assigns $f^* = -13.75$ and $x^* = (0, 2, 3, 0.5)$.

The remaining search always leads either to an infeasible relaxation or to a relaxation with an objective value worse than f^*.



The final solution: $x^* = (0, 2, 3, 0.5)$ and $f^* = -13.75$.

Cutting Planes

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Another strategy might be to successively cut out non-integer optimal solutions and preserve the integer ones until an integer optimal solution is computed by the LP relaxation

Removing Non-Integer Solutions

The basic branch and bound method generates two new problems in every step.

Another strategy might be to successively cut out non-integer optimal solutions and preserve the integer ones until an integer optimal solution is computed by the LP relaxation

We consider a concrete method for obtaining such cuts from the ILP constraints called *Gomory cuts*.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z} \text{ for } x \in \mathcal{D} \end{aligned}$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z} \text{ for } x \in \mathcal{D} \end{aligned}$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

We demand the integer solution only for the original $\mathcal{D}$ variables.

Gomory Cuts

Consider an ILP and transform it into a MILP by adding slack variables:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z} \text{ for } x \in \mathcal{D} \end{aligned}$$

Here, $\mathcal{D}$ contains the original (i.e., non-slack) variables of the ILP.

We demand the integer solution only for the original $\mathcal{D}$ variables.

However, one can prove that if all constants in the ILP are integer, then there is an optimal solution where all variables (including the slacks) are integer-valued.

Gomory Cuts

Let $A = (u_1 \ \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ \ldots, u_m)$.

Gomory Cuts

Let $A = (u_1 \ \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ \ldots, u_m)$.

Consider the canonical tableau for B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

The $-z$ row is omitted as it is unnecessary for the discussion.

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

Gomory Cuts

Let $A = (u_1 \ldots, u_n)$, the basis $\{x_1, \ldots, x_n\}$, $B = (u_1 \ldots, u_m)$.

Consider the canonical tableau for B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

The $-z$ row is omitted as it is unnecessary for the discussion.

$$u_k = B(y_{1k}, \ldots, y_{mk})^\top \text{ for } k = 1, \ldots, n \text{ and } b' = B^{-1}b$$

Consider a basic solution $x = (b'_1, \ldots, b'_m, 0, \ldots, 0)$.

If all $b'_1, \ldots, b'_m$ are integers, then also x solves the ILP.

Otherwise, assume that b'_i is not an integer.

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b_i'$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)} x_{m+1} + \cdots + y_{in} x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b'_i \rfloor$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)}x_{m+1} + \cdots + y_{in}x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b'_i \rfloor$$

But, subtracting the inequalities, integer feasible solutions x satisfy:

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts

From the tableau, we know that every feasible solution x satisfies:

$$x_i + y_{i(m+1)} x_{m+1} + \cdots + y_{in} x_n = b'_i$$

Then, x also satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq b'_i$$

Moreover, any *integer feasible solution* x satisfies:

$$x_i + \lfloor y_{i(m+1)} \rfloor x_{m+1} + \cdots + \lfloor y_{in} \rfloor x_n \leq \lfloor b'_i \rfloor$$

But, subtracting the inequalities, integer feasible solutions x satisfy:

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b'_i - \lfloor b'_i \rfloor$$

But note that the *basic feasible solution* $x = (b'_1, \ldots, b'_m, 0, \ldots, 0)$ *does not* satisfy the last inequality because $b'_i > \lfloor b'_i \rfloor$ and $x_{m+1} = \cdots = x_n = 0$.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.

Assume that the basic solution x w.r.t. B is non-integer.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor)x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor)x_n \geq b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Add the Gomory cut and the constraint $x_{n+1} \geq 0$ to the program.

Gomory Cuts Method

Assume that we have solved the LP and reached a basis of B.
Assume that the basic solution x w.r.t. B is non-integer.

Consider the canonical tableau for the basis B:

$$A' = \begin{pmatrix} 1 & \cdots & 0 & y_{1(m+1)} & \cdots & y_{1n} & b'_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & y_{m(m+1)} & \cdots & y_{mn} & b'_m \end{pmatrix}$$

Choose a non-integer component $x_i = b'_i$ of the basic feasible solution w.r.t. B and consider the constraint

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n \geq b'_i - \lfloor b'_i \rfloor$$

Transform the above inequality into equality by introducing a new variable x_{n+1} and obtain the following constraint (*Gomory cut*)

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

Add the Gomory cut and the constraint $x_{n+1} \geq 0$ to the program.

Repeat until an integer solution is reached.

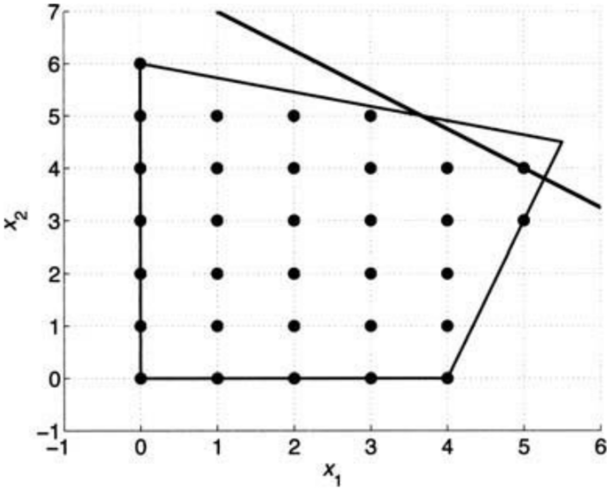
Example

Consider ILP:

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 \leq 12 \\ & 3x_1 + 11x_2 \leq 66 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$

Adding slack variables x_3, x_4 we obtain the following MILP:

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$



7
6
5
4
3
2
1
0
−1
x_2
−1
0
1
2
3
4
5
6
x_1

We have

$$\begin{aligned}
\text{minimize} \quad & -3x_1 - 4x_2 \\
\text{subject to} \quad & 3x_1 - x_2 + x_3 = 12 \\
& 3x_1 + 11x_2 + x_4 = 66 \\
& x_1, x_2, x_3, x_4 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}$$

We have

$$\begin{array}{rl} \text{minimize} & -3x_1 - 4x_2 \\ \text{subject to} & 3x_1 - x_2 + x_3 = 12 \\ & 3x_1 + 11x_2 + x_4 = 66 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array}$$

An optimal basic solution to the LP relaxation is

$$\left(\frac{11}{2}, \frac{9}{2}, 0, 0\right)^\top$$

and the canonical tableau w.r.t. the basis $\{x_1, x_2\}$ is

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & b' \\ 1 & 0 & \frac{11}{36} & \frac{1}{36} & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & \frac{9}{2} \end{pmatrix}$$

Let us introduce the Gomory cut corresponding to the variable x_1.

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & b' \\ 1 & 0 & \frac{11}{36} & \frac{1}{36} & \frac{11}{2} \\ 0 & 1 & -\frac{1}{12} & \frac{1}{12} & \frac{9}{2} \end{pmatrix}$$

Then

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

with $i = 1$ and $m = 2$ turns into

$$\left(\frac{11}{36} - 0\right) x_3 + \left(\frac{1}{36} - 0\right) x_4 - x_5 = \frac{1}{2} \quad (= \frac{11}{2} - 5)$$

We add this constraint to our MILP.

$$
\begin{aligned}
\text{minimize} \quad & -3x_1 - 4x_2 \\
\text{subject to} \quad & 3x_1 - x_2 + x_3 = 12 \\
& 3x_1 + 11x_2 + x_4 = 66 \\
& \tfrac{11}{36}x_3 + \tfrac{1}{36}x_4 - x_5 = \tfrac{1}{2} \\
& x_1, x_2, x_3, x_4 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}
$$

Solving the LP relaxation yields

$$\left(5, \frac{51}{11}, \frac{18}{11}, 0, 0\right)^\top$$

The canonical tableau for the solution is

$$
\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 & b' \\
1 & 0 & 0 & 0 & 1 & 5 \\
0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & \frac{51}{11} \\
0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & \frac{18}{11}
\end{pmatrix}
$$

Introduce the Gomory cut for x_2.

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & b' \\ 1 & 0 & 0 & 0 & 1 & 5 \\ 0 & 1 & 0 & \frac{1}{11} & -\frac{3}{11} & \frac{51}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -\frac{36}{11} & \frac{18}{11} \end{pmatrix}$$

Then

$$(y_{i(m+1)} - \lfloor y_{i(m+1)} \rfloor) x_{m+1} + \cdots + (y_{in} - \lfloor y_{in} \rfloor) x_n - x_{n+1} = b'_i - \lfloor b'_i \rfloor$$

with $i = 2$ and $m = 3$ turns into

$$\left(\frac{1}{11} - 0\right) x_4 + \left(-\frac{3}{11} + \frac{11}{11}\right) x_5 - x_6 = \frac{7}{11} \quad (= \frac{51}{11} - \frac{44}{11})$$

We add this to our MILP.

$$
\begin{array}{rl}
\text{minimize} & -3x_1 - 4x_2 \\
\text{subject to} & 3x_1 - x_2 + x_3 = 12 \\
& 3x_1 + 11x_2 + x_4 = 66 \\
& \frac{11}{36}x_3 + \frac{1}{36}x_4 - x_5 = \frac{1}{2} \\
& \frac{1}{11}x_4 + \frac{8}{11}x_5 - x_6 = \frac{7}{11} \\
& x_1, x_2, x_3, x_4 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{array}
$$

Once more the solution of the above is non-integer. However, introducing another Gomory cut (and a variable x_7) would yield a solution:

$$(5, 4, 1, 7, 0, 0, 0)^\top$$

Which gives the point $(x_1, x_2) = (5, 4)$ corresponding to the graphical solution.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting planes are also used in other non-linear, non-smooth optimization methods.

Cutting Planes Technique

The method based on Gomory cuts was one of the first solutions to the integer linear programming problem with proven convergence (in the 1950s).

The convergence rate is unsatisfactory in practice; many more methods have been devised based on algebraic principles (combinations of inequalities and rounding), geometry, etc.

Cutting planes are also used in other non-linear, non-smooth optimization methods.

Most importantly, cutting plane techniques are combined with branch and bound methods. The constraints are introduced before branching to eliminate some solutions before the split.

The resulting method is called *branch and cut*.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.
- Integer Linear Programming (ILP)
  Linear objective and constraints. All variables restricted to $\mathbb{Z}$.

Summary of Integer Linear Programming

We have considered:

- Linear Programming (LP)
  Linear objective and constraints.
- 0-1 Integer Linear Programming (0-1 ILP)
  Linear objective and constraints. All variables restricted to $\{0, 1\}$.
- 0-1 Mixed Integer Programming (0-1 MILP)
  Linear objective and constraints. Some variables restricted to $\{0, 1\}$.
- Integer Linear Programming (ILP)
  Linear objective and constraints. All variables restricted to $\mathbb{Z}$.
- Mixed Integer Linear Programming (MILP)
  Linear objective and constraints. Some variables restricted to $\mathbb{Z}$.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.

  Linear programming is in $\mathbb{P}$-time.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.
- Cutting planes
  - Sequentially cut out portions of the LP relaxation feasible space by introducing cuts based on solutions of LP relaxations.

Summary of Integer Linear Programming

Complexity:

- Even the 0-1 Integer Linear Programming is NP-hard.
  Linear programming is in $\mathbb{P}$-time.

Algorithms:

- Branch and Bound
  - 0-1 MILP: Search through possible assignments of 0 and 1 to some discrete variables while solving the LP relaxations
    Branching with the choice of 0/1 values of variables, bounding with a solution found so far.
  - MILP: Solve LP relaxation, use non-integer values of the solution to introduce constraints, removing such values from the solution.
- Cutting planes
  - Sequentially cut out portions of the LP relaxation feasible space by introducing cuts based on solutions of LP relaxations.
  - Does not branch but is usually combined with branch and bound (branch and cut).

Gradient-Free Optimization

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

What if the functions are just black boxes that can be evaluated but nothing else?

Gradient-Free Methods

So far, we have explored problems where the objective f and the constraint functions h_j, g_i are known and (at least) differentiable.

What if the functions are just black boxes that can be evaluated but nothing else?

What if the evaluation itself is costly?

Example: GPU parameters fine-tunning:

- Tens of parameters.
- The objective is to execute GPU software as efficiently as possible (tested by execution of a benchmark software suite)
- Evaluation of the objective function = Execution of a benchmark software suite
- How do we optimize the parameters?

Nothing is (possibly) differentiable here. Small changes in the parameters may give wildly different results.

There are many methods for such optimization. Most of them, of course, are without any convergence and efficiency guarantees.

Gradient-Free Methods Zoo

	Search		Algorithm		Function evaluation		Stochasticity	
	Local	Global	Mathematical	Heuristic	Direct	Surrogate	Deterministic	Stochastic
Nelder–Mead	•			•	•		•	
GPS		•	•		•		•	
MADS		•	•		•			•
Trust region	•		•			•	•	
Implicit filtering	•		•			•	•	
DIRECT		•	•		•		•	
MCS		•	•		•		•	
EGO		•	•			•	•	
Hit and run		•		•	•			•
Evolutionary		•		•	•			•

For more details see "Engineering Design Optimization" by Joaquim R. R. A. Martins and Andrew Ning

Evolutionary

"Evolutionary algorithms are inspired by processes that occur in nature or society. There is a plethora of evolutionary algorithms in the literature, thanks to the fertile imagination of the research community and a never-ending supply of phenomena for inspiration."

Evolutionary

"Evolutionary algorithms are inspired by processes that occur in nature or society. There is a plethora of evolutionary algorithms in the literature, thanks to the fertile imagination of the research community and a never-ending supply of phenomena for inspiration."

ant colony optimization, bee colony algorithm, fish swarm, artificial flora optimization algorithm, bacterial foraging optimization, bat algorithm, big bang–big crunch algorithm, biogeography-based optimization, bird mating optimizer, cat swarm, cockroach swarm, cuckoo search, design by shopping paradigm, dolphin echolocation algorithm, elephant herding optimization, firefly algorithm, flower pollination algorithm, fruit fly optimization algorithm, galactic swarm optimization, gray wolf optimizer, grenade explosion method, harmony search algorithm, hummingbird optimization algorithm, hybrid glowworm swarm optimization algorithm, imperialist competitive algorithm, intelligent water drops, invasive weed optimization, mine bomb algorithm, monarch butterfly optimization, moth-flame optimization algorithm, penguin search optimization algorithm, quantum-behaved particle swarm optimization, salp swarm algorithm, teaching–learning-based optimization, whale optimization algorithm, and water cycle algorithm, ...

Two Methods

To appreciate the gradient-free approaches, we shall (rather arbitrarily) concentrate on two methods:

- Nelder-Mead
- Particle Swarm Optimization

Both methods are somehow biologically motivated.

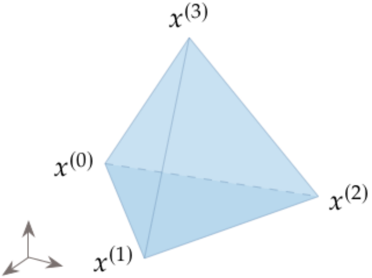
We consider the unconstrained optimization. That is, assume an objective function $f : \mathbb{R}^n \to \mathbb{R}$.

Nelder-Mead

The Nelder-Mead algorithm is based on a *simplex* defined by a set of $n+1$ points in $\mathbb{R}^n$:

$$X = \left\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\right\} \subseteq \mathbb{R}^n$$

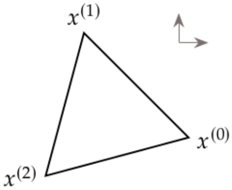
In two dimensions, the simplex is a triangle, and in three dimensions, it becomes a tetrahedron



A minimizer is approximated by a simplex node with a minimum value of f. The simplex changes in every step.

Nelder-Mead

Initially, $n+1$ nodes of the simplex need to be chosen: Typically, equal-length of edges and $x^{(0)}$ will be our starting point x_0.



$$x^{(i)} = x^{(0)} + s^{(i)},$$

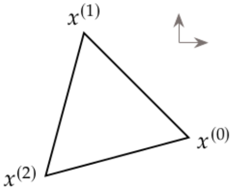
where $s^{(i)}$ is a vector whose components j are defined by

$$s_j^{(i)} = \begin{cases} \frac{L}{n\sqrt{2}}(\sqrt{n+1} - 1) + \frac{L}{\sqrt{2}}, & \text{if } j = i \\ \frac{L}{n\sqrt{2}}(\sqrt{n+1} - 1), & \text{if } j \neq i. \end{cases}$$

Here, L is the length of each side.

Nelder-Mead

Initially, $n+1$ nodes of the simplex need to be chosen: Typically, equal-length of edges and $x^{(0)}$ will be our starting point x_0.



$$x^{(i)} = x^{(0)} + s^{(i)},$$

where $s^{(i)}$ is a vector whose components j are defined by

$$s_j^{(i)} = \begin{cases} \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1) + \frac{L}{\sqrt{2}}, & \text{if } j = i \\ \frac{L}{n\sqrt{2}}(\sqrt{n+1}-1), & \text{if } j \neq i. \end{cases}$$

Here, L is the length of each side.

Nelder-Mead method proceeds by modifying the simplex so that the values of f in the vertices (hopefully) decrease.

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection*, *expansion*, *outside contraction*, *inside contraction*, and *shrinking*.

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection, expansion, outside contraction, inside contraction, and shrinking*.

Except for shrinking, each operation generates a new point,

$$x = x_c + \alpha \left(x_c - x^{(n)} \right),$$

Here $\alpha \in \mathbb{R}$ and x_c is the centroid of all the points except for the worst one, that is, assuming $x^{(n)}$ maximizes f among the nodes

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

Nelder-Mead

The Nelder-Mead algorithm performs five main operations on the simplex to create a new one: *reflection, expansion, outside contraction, inside contraction, and shrinking*.

Except for shrinking, each operation generates a new point,

$$x = x_c + \alpha \left(x_c - x^{(n)} \right),$$

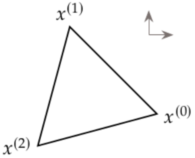
Here $\alpha \in \mathbb{R}$ and x_c is the centroid of all the points except for the worst one, that is, assuming $x^{(n)}$ maximizes f among the nodes

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

This generates a new point along the line that connects the worst point, $x^{(n)}$, and the centroid of the remaining points, x_c.

This direction can be seen as a possible descent direction.

Nelder-Mead Algorithm



1. Start with a simplex $x^{(0)}, \ldots, x^{(n)}$

Assume an order of these points:

$$f(x^{(0)}) \leq \ldots \leq f(x^{(n)})$$

2. Calculate the centroid

$$x_c = \frac{1}{n} \sum_{i=0}^{n-1} x^{(i)}$$

Nelder-Mead Algorithm (Reflection)

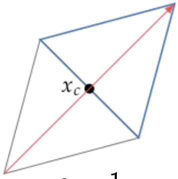
3. **Reflection** of $x^{(n)}$ over the centroid:

$$x_r = x_c + \alpha \left(x_c - x^{(n)} \right) \quad \text{for } \alpha > 0$$

If $f(x^{(0)}) \leq f(x_r) < f(x^{(n-1)})$, then

Replace $x^{(n)}$ with x_r

Go to 1.



$\alpha = 1$

Now going further we know that either $f(x_r) < f(x^{(0)})$, or $f(x_r) \geq f(x^{(n-1)})$

Nelder-Mead Algorithm (Expansion)

4. **Expansion**

   If $f(x_r) < f(x^{(0)})$, then

   Compute

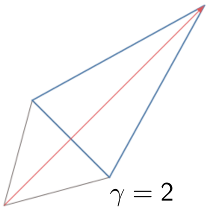
   $$x_e = x_c + \gamma\left(x_c - x^{(n)}\right) \quad \text{for } \gamma > 1$$

   If $f(x_e) < f(x_r)$, then

   Replace $x^{(n)}$ with x_e.

   Else, replace $x^{(n)}$ with x_r.

   Go to 1.



Now going further we know that $f(x_r) \geq f(x^{(n-1)})$

Nelder-Mead (Contraction)

5. **Contraction**

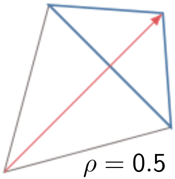
If $f(x_r) < f(x^{(n)})$, then compute outside contraction

$$x_{oc} = x_c + \rho\,(x_r - x_c) \qquad \text{for } 0 < \rho \leq 0.5$$

If $f(x_{oc}) < f(x_r)$, then

Replace $x^{(n)}$ with x_{oc}

Go to 1.



$\rho = 0.5$

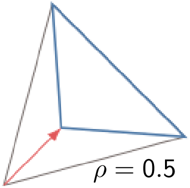
If $f(x_r) \geq f(x^{(n)})$, then compute inside contraction

$$x_{ic} = x_c + \rho\left(x^{(n)} - x_c\right) \qquad \text{for } 0 < \rho \leq 0.5$$

If $f(x_{ic}) < f(x^{(n)})$, then

Replace $x^{(n)}$ with x_{ic}

Go to 1.



$\rho = 0.5$

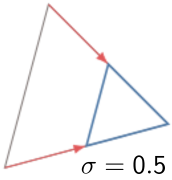
Nelder-Mead (Shrink)

6. **Shrink**

   Replace all points $x^{(k)}$ for $k > 0$ with

   $$x^{(k)} = x^{(k)} + \sigma(x^{(k)} - x^{(0)}) \quad \text{for } 0 < \sigma < 1$$

   Go to 1.



Nelder-Mead

The above procedure is repeated until convergence. This may be decided, e.g., based on the size of the simplex:

$$\Delta_x = \sum_{i=0}^{n-1} \left\| x^{(i)} - x^{(n)} \right\| < \epsilon$$

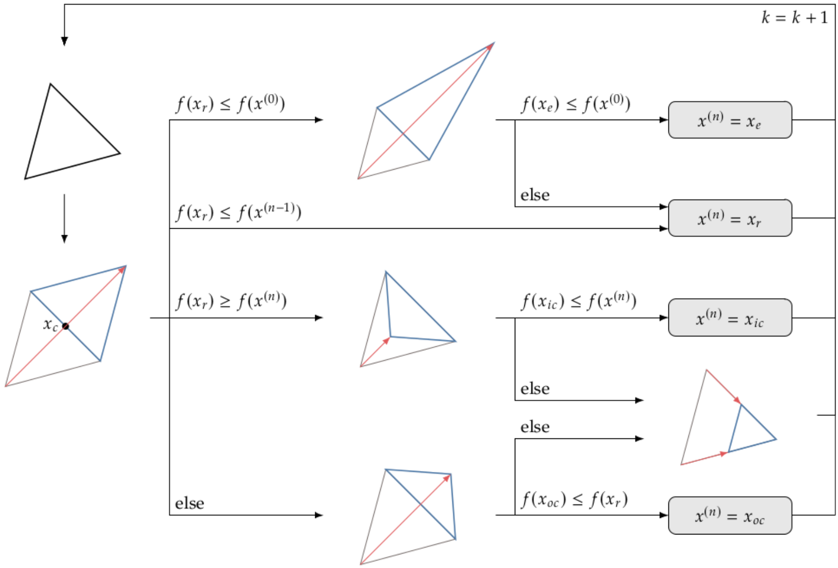
Nelder-Mead

The above procedure is repeated until convergence. This may be decided, e.g., based on the size of the simplex:

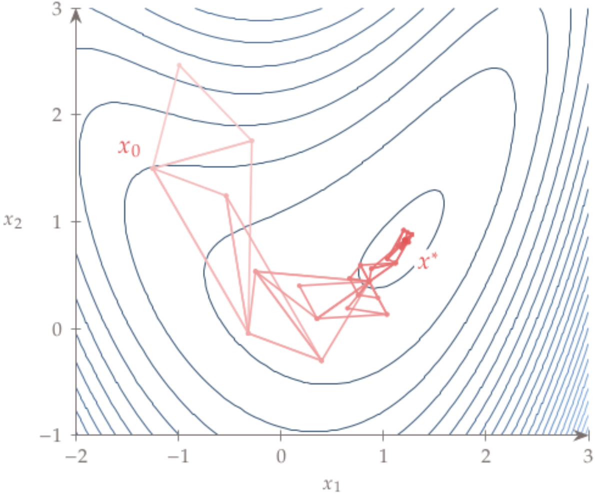
$$\Delta_x = \sum_{i=0}^{n-1} \left\| x^{(i)} - x^{(n)} \right\| < \epsilon$$

Standard values for constants are:

- Reflection $\alpha = 1$
- Expansion $\gamma = 2$
- Contraction $\rho = 0.5$
- Shrink $\sigma = 0.5$


$k = k + 1$
$f(x_r) \le f(x^{(0)})$
$f(x_e) \le f(x^{(0)})$
$x^{(n)} = x_e$
else
$f(x_r) \le f(x^{(n-1)})$
$x^{(n)} = x_r$
x_c
$f(x_r) \ge f(x^{(n)})$
$f(x_{ic}) \le f(x^{(n)})$
$x^{(n)} = x_{ic}$
else
else
else
$f(x_{oc}) \le f(x_r)$
$x^{(n)} = x_{oc}$

Nelder-Mead Example



Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.
- This velocity changes according to the past objective function values of that particle and the current objective values of the rest of the particles.

Particle Swarm Optimization

- The “swarm” in PSO is a set of points (agents or particles) that move in space, looking for the best solution.
- Each particle moves according to its velocity.
- This velocity changes according to the past objective function values of that particle and the current objective values of the rest of the particles.
- Each particle remembers the point where it found its best result so far, and it exchanges the information with the swarm.

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

- The first term is momentum.

  α is usually set from the interval $[0.8, 1.2]$, higher α motivates exploration, smaller α convergence towards (a local) minimizer.

PSO

The position of particle i for iteration $k+1$ is updated according to

$$x_{k+1}^{(i)} = x_k^{(i)} + v_{k+1}^{(i)} \Delta t,$$

Where Δt is a constant artificial time step. The velocity for each particle is updated as follows:

$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}$$

- The first term is momentum.
  α is usually set from the interval $[0.8, 1.2]$, higher α motivates exploration, smaller α convergence towards (a local) minimizer.
- $x_{\text{best}}^{(i)}$ is the first minimum objective point visited by the i-th particle.
  β is usually set randomly from $[0, \beta_{\max}]$. $\beta_{\max}$ is usually selected from the interval $[0, 2]$, closer to 2.
- x_{best} is a minimum objective point visited by any particle.
  γ is also usually set randomly from the interval $[0, \gamma_{\max}]$. $\gamma_{\max}$ is usually selected from the interval $[0, 2]$, closer to 2.

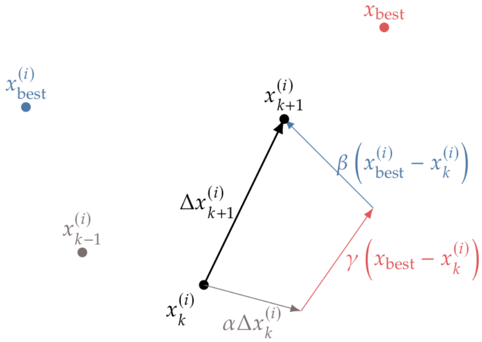
$$v_{k+1}^{(i)} = \alpha v_k^{(i)} + \beta \frac{x_{\text{best}}^{(i)} - x_k^{(i)}}{\Delta t} + \gamma \frac{x_{\text{best}} - x_k^{(i)}}{\Delta t}.$$

Eliminate Δt by multiplying with Δt:

$$\Delta x_{k+1}^{(i)} = \alpha \Delta x_k^{(i)} + \beta \left(x_{\text{best}}^{(i)} - x_k^{(i)} \right) + \gamma \left(x_{\text{best}} - x_k^{(i)} \right)$$

Then, update the particle position for the next iteration:

$$x_{k+1}^{(i)} = x_k^{(i)} + \Delta x_{k+1}^{(i)}.$$



PSO

- Initialization is usually done randomly.

PSO

- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.

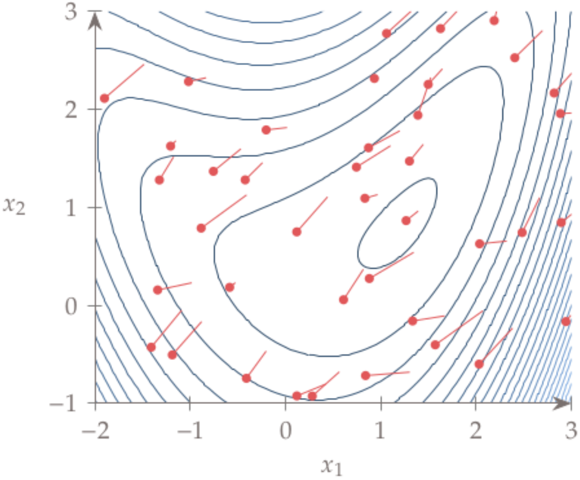
PSO

- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.
- It is also helpful to impose a maximum velocity. Otherwise, updates completely unrelated to the previous positions might be made.

PSO

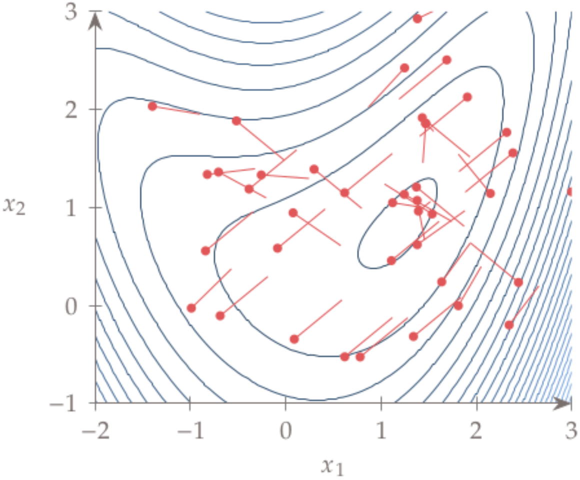
- Initialization is usually done randomly.
- The particles should stay in a bounded region. When a particle wants to leave the region, reorient the velocity or reset the position of the particle.
- It is also helpful to impose a maximum velocity. Otherwise, updates completely unrelated to the previous positions might be made.
- The velocity may be decreased gradually to exchange exploitation with exploration.

Example



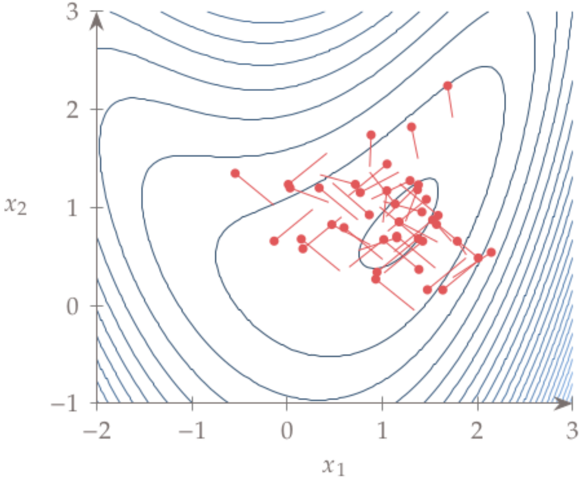
$K = 0$

Example



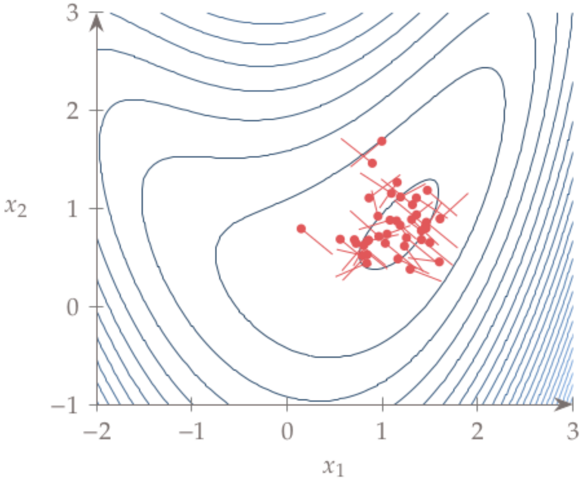
$K = 1$

Example



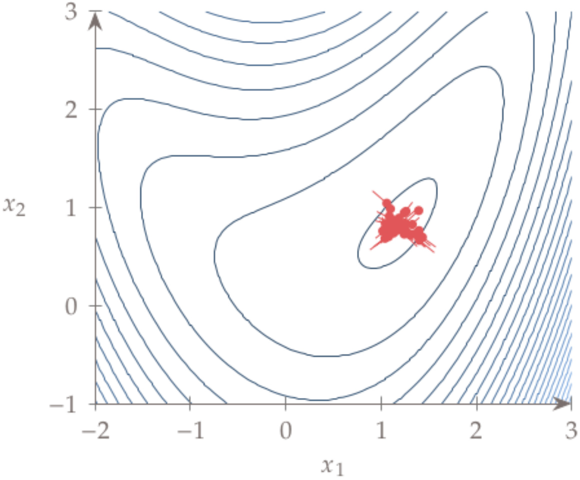
$K = 3$

Example



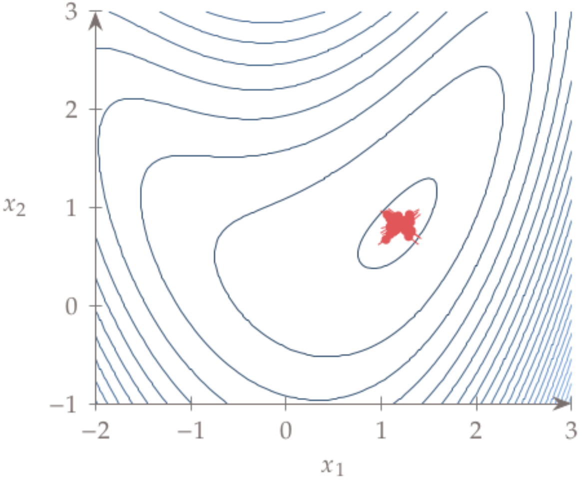
$K = 5$

Example



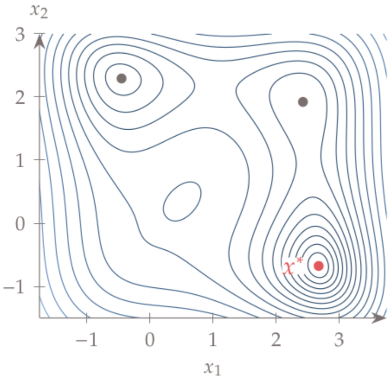
$K = 12$

Example



$K = 17$

Jones Function



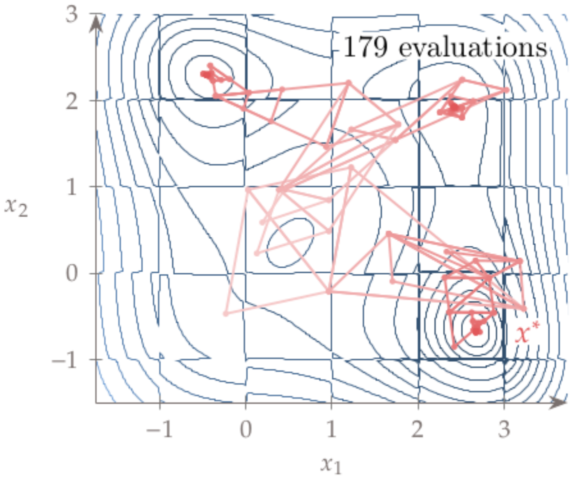
$$f(x_1, x_2) = x_1^4 + x_2^4 - 4x_1^3 - 3x_2^3 + 2x_1^2 + 2x_1x_2$$

Global minimum: $f(x^*) = -13.5320$ at $x^* = (2.6732, -0.6759)$.

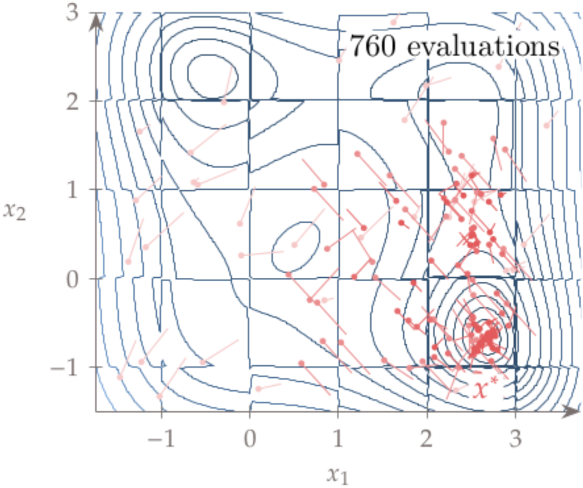
Local minima: $f(x) = -9.7770$ at $x = (-0.4495, 2.2928)$

$f(x) = -9.0312$ at $x = (2.4239, 1.9219)$

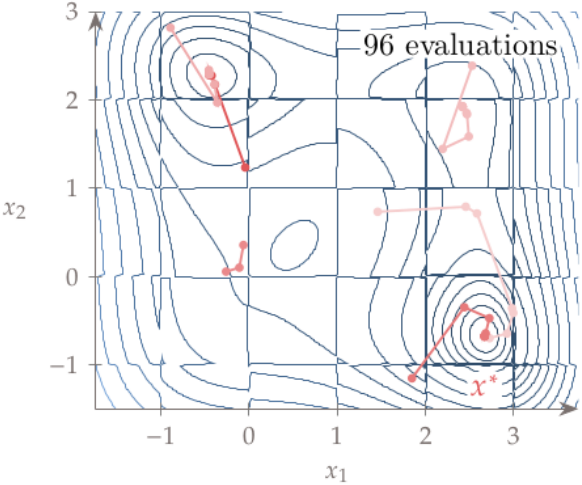
Make it discontinuous by adding $4 \lceil \sin(\pi x_1) \sin(\pi x_2) \rceil$



Nelder-Mead: 179 evaluations were needed to reach the minimum (with restarts due to local minima).



Particle Swarm Optimization: 760 evaluations found the global minimum without restarts.



Quasi-Newton with restarts: 96 evaluations needed. Converged in two out of six random restarts.

FINALE!

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming
- Linear Objective and Constraints
  - Simplex Method (including degenerate case and tableaus)
  - Branch & Bound
  - Gomory Cuts

Summary

We have considered the following methods:

- Unconstrained & Differentiable Objective
  - Line Search with step size determined by Wolfe conditions and direction determined by
    - Gradient Descent
    - Newton's Method (2nd derivatives needed)
    - Quasi-Newton: SR-1, BFGS
- Constrained & Differentiable Objective
  - Penalty Methods
    - Exterior (quadratic penalty)
    - Interior/Barrier (inverse and logarithmic barriers)
  - Lagrangian used in Sequential Quadratic Programming
- Linear Objective and Constraints
  - Simplex Method (including degenerate case and tableaus)
  - Branch & Bound
  - Gomory Cuts
- Unconstrained & Non-Differentiable (just a few examples)
  - Nelder-Mead
  - Particle Swarm Optimization

Most Notable Omissions

- Conjugate Gradient Methods

  Unfortunately, I had to choose between quasi-Newton and CG.
- Trust Region Methods
- Combinatorial, Multiobjective, Stochastic, Bayesian (etc.) Optimization

  Completely different areas with different methods.
- Infinitely many non-differentiable optimization methods motivated by arbitrary phenomena from:
  - biology
  - chemistry
  - physics
  - economics
  - politics
  - mathematics
  - agriculture
  - pop-culture
  - Scientology
  - astrology
  - ... ...