

# One-Phase Simplex Algorithm (1947)

```
Maximize(LPinstance (A, b, c)){
  {The instance must be in standard form and origin-feasible}
  Construct the feasible dictionary  $D$  for the instance;
  while(some  $x_i$  has positive coefficient in eq. for  $z$ ){
    Find the variable  $x_j$  which constrains increasing  $x_i$ ;
     $D := D[x_i \leftrightarrow x_j]$ ;
  }
  return solution corresponding to  $D$ ;
}
```

## Design:

- 1 How to find first feasible solution. *Done for case of feasible origin.*
- 2 Define neighborhood structure. *Done - Pivoting.*
- 3 Strategy for choosing neighbor. *Done - Bland's rule.*

## Analysis:

- 1 Partial correctness (Termination  $\Rightarrow$  Correctness). *Done.*
- 2 Termination. *Done - Bland's rule*
- 3 Complexity.

# Solving non-origo feasible standard instances

**Maximize**  $\sum_{j=1}^n c_j x_j$  **subject to**

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 1, 2, \dots, n$$

# Solving non-origo feasible standard instances

**Maximize**  $\sum_{j=1}^n c_j x_j$  **subject to**

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 1, 2, \dots, n$$

Assume  $b_1 < 0$  and  $\forall i : b_1 \leq b_i$ .

# Solving non-origo feasible standard instances

**Maximize**  $\sum_{j=1}^n c_j x_j$  **subject to**

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 1, 2, \dots, n$$

Assume  $b_1 < 0$  and  $\forall i : b_1 \leq b_i$ .

Generate **Auxillary Program: Minimize**  $x_0$  **subject to**

$$\left(\sum_{j=1}^n a_{ij} x_j\right) - x_0 \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 0, 1, 2, \dots, n$$

# Auxillary Program

**Minimize**  $x_0$  **subject to**

$$\left(\sum_{j=1}^n a_{ij}x_j\right) - x_0 \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 0, 1, 2, \dots, n$$

**Minimize**  $x_0$  **subject to**

$$\left(\sum_{j=1}^n a_{ij}x_j\right) - x_0 \leq b_i, \quad i = 1, 2, \dots, m$$
$$x_j \geq 0, \quad j = 0, 1, 2, \dots, n$$

**Crucial property:** The auxillary instance has optimal value  $x_0 = 0$  iff the original instance is feasible. If so, a solution with this value is also a feasible solution to the original instance.

A feasible solution for the auxillary instance is

- $x_0 = -b_1,$
- $x_1 = x_2 = \dots = x_n = 0.$

## Dictionary for auxillary instance

$$x_{n+i} = b_i - \sum_{j=1}^n a_{ij}x_j + x_0, \quad i = 1, \dots, m$$
$$w = -x_0$$

Dictionary infeasible but pivoting on  $x_0$  and  $x_{n+1}$  yields feasible dictionary corresponding to

- $x_0 = -b_1$ ,
- $x_1 = x_2 = \dots = x_n = x_{n+1} = 0$  and with
- $x_{n+j} = b_j - b_1 \geq 0$  for  $j \geq 2$ .

# Example

**Maximize**  $x_1 - x_2 + x_3$  **Subject to**

$$2x_1 - 3x_2 + x_3 \leq -5$$

$$4x_1 - x_2 + 2x_3 \leq 4$$

$$-x_1 + x_2 - 2x_3 \leq -1$$

$$x_1, x_2, x_3 \geq 0$$

# Example

**Maximize**  $x_1 - x_2 + x_3$  **Subject to**

$$2x_1 - 3x_2 + x_3 \leq -5$$

$$4x_1 - x_2 + 2x_3 \leq 4$$

$$-x_1 + x_2 - 2x_3 \leq -1$$

$$x_1, x_2, x_3 \geq 0$$

Auxiliary problem:

**Minimize**  $x_0$  **Subject to**

$$2x_1 - 3x_2 + x_3 - x_0 \leq -5$$

$$4x_1 - x_2 + 2x_3 - x_0 \leq 4$$

$$-x_1 + x_2 - 2x_3 - x_0 \leq -1$$

$$x_0, x_1, x_2, x_3 \geq 0$$

# Dictionary for auxiliary program

Maximize  $w$  subject to  $x_0, x_1, x_2, \dots, x_6 \geq 0$  and

$$\begin{aligned}x_4 &= -5 - 2x_1 + 3x_2 - x_3 + x_0 \\x_5 &= 4 - 2x_1 + x_2 - 2x_3 + x_0 \\x_6 &= -1 + x_1 - x_2 + 2x_3 + x_0 \\w &= \phantom{-5 - 2x_1 + 3x_2 - x_3 + x_0} - x_0\end{aligned}$$

# Dictionary for auxiliary program

Maximize  $w$  subject to  $x_0, x_1, x_2, \dots, x_6 \geq 0$  and

$$\begin{aligned}x_4 &= -5 - 2x_1 + 3x_2 - x_3 + x_0 \\x_5 &= 4 - 2x_1 + x_2 - 2x_3 + x_0 \\x_6 &= -1 + x_1 - x_2 + 2x_3 + x_0 \\w &= -x_0\end{aligned}$$

Pivoting  $x_0$  into the basis and  $x_4$  out of the basis:

$$\begin{aligned}x_0 &= 5 + 2x_1 - 3x_2 + x_3 + x_4 \\x_5 &= 9 - 2x_2 - x_3 + x_4 \\x_6 &= 4 + 3x_1 - 4x_2 + 3x_3 + x_4 \\w &= -5 - 2x_1 + 3x_3 - x_3 - x_4\end{aligned}$$

# The two-phase simplex algorithm

Apply the simplex algorithm to the auxiliary instance.

# The two-phase simplex algorithm

Apply the simplex algorithm to the auxiliary instance.

- If optimal solution has value  $w < 0$ , the original instance is INFEASIBLE.

# The two-phase simplex algorithm

Apply the simplex algorithm to the auxiliary instance.

- If optimal solution has value  $w < 0$ , the original instance is INFEASIBLE.
- If optimal solution has value  $w = -x_0 = 0$ , the solution is a solution to the original instance.

# The two-phase simplex algorithm

Apply the simplex algorithm to the auxiliary instance.

- If optimal solution has value  $w < 0$ , the original instance is INFEASIBLE.
- If optimal solution has value  $w = -x_0 = 0$ , the solution is a solution to the original instance.
- If  $x_0$  is in the basis make a pivot putting it out of the basis (as  $x_0 = 0$ , this does not change the solution).

# The two-phase simplex algorithm

Apply the simplex algorithm to the auxiliary instance.

- If optimal solution has value  $w < 0$ , the original instance is INFEASIBLE.
- If optimal solution has value  $w = -x_0 = 0$ , the solution is a solution to the original instance.
- If  $x_0$  is in the basis make a pivot putting it out of the basis (as  $x_0 = 0$ , this does not change the solution).
- Removing all terms involving  $x_0$  from the dictionary now yields a feasible dictionary for the original instance, because..

## Final dictionary for auxillary instance

$$x_i = b_i - \sum_j \bar{a}_{ij} x_j + \alpha_i x_0, \quad i \in B$$

Non-negative solutions corresponds exactly to solutions of:

$$\left( \sum_{j=1}^n a_{ij} x_j \right) - x_0 \leq b_i, \quad i = 1, 2, \dots, m$$

## Restricting to $x_0 = 0$ :

$$x_i = b_i - \sum_j \bar{a}_{ij} x_j, \quad i \in B$$

Non-negative solutions corresponds exactly to solutions of:

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, 2, \dots, m$$

# Final dictionary for first phase in example

$$\begin{aligned}x_3 &= 1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6 - 0.8x_0 \\x_2 &= 2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6 - 0.6x_0 \\x_5 &= 3 - x_1 - x_6 + 2x_0 \\w &= -x_0\end{aligned}$$

## Final dictionary for first phase in example

$$\begin{aligned}x_3 &= 1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6 - 0.8x_0 \\x_2 &= 2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6 - 0.6x_0 \\x_5 &= 3 - x_1 - x_6 + 2x_0 \\w &= -x_0\end{aligned}$$

Removing  $x_0$ :

$$\begin{aligned}x_3 &= 1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6 \\x_2 &= 2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6 \\x_5 &= 3 - x_1 - x_6\end{aligned}$$

## Final dictionary for first phase in example

$$\begin{aligned}x_3 &= 1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6 - 0.8x_0 \\x_2 &= 2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6 - 0.6x_0 \\x_5 &= 3 - x_1 - x_6 + 2x_0 \\w &= -x_0\end{aligned}$$

Removing  $x_0$ :

$$\begin{aligned}x_3 &= 1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6 \\x_2 &= 2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6 \\x_5 &= 3 - x_1 - x_6\end{aligned}$$

Last row:  $z = x_1 - x_2 + x_3 = x_1 - (2.2 + 0.6x_1 + 0.4x_4 + 0.2x_6) + (1.6 - 0.2x_1 + 0.2x_4 + 0.6x_6) = -0.6 + 0.2x_1 - 0.2x_4 + 0.4x_6$

# Fundamental theorem of linear programming

For every linear program in standard form:

- 1 If it has no optimal solution, it is either infeasible or unbounded.
- 2 If it has a feasible solution, then it has a basic feasible solution.
- 3 If it has an optimal solution, then it has a basic optimal solution.

# Fundamental theorem of linear programming

For every linear program in standard form:

- 1 If it has no optimal solution, it is either infeasible or unbounded.
- 2 If it has a feasible solution, then it has a basic feasible solution.
- 3 If it has an optimal solution, then it has a basic optimal solution.

Proof: The two-phase simplex algorithm!

Complexity of Local search  $\approx$

Time for one improve  $\times$

Number of improve steps.

- Identify variable  $x_i$  and  $x_j$  to switch place.
- Rewrite expression for  $x_j$  as an expression for  $x_i$
- Substitute this expression for  $x_j$  in all other equations in  $D$
- **Return** resulting system of equations.

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis:

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis:

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis: Time  $O(m)$ .

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis: Time  $O(m)$ .

Rewrite equation for  $x_j$  to equation for  $x_i$ :

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis: Time  $O(m)$ .

Rewrite equation for  $x_j$  to equation for  $x_i$ : Time  $O(n)$ .

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis: Time  $O(m)$ .

Rewrite equation for  $x_j$  to equation for  $x_i$ : Time  $O(n)$ .

Substitute equation for  $x_j$  everywhere:

# Time for one improvement

Assume dictionaries represented as arrays.

Identify the variable  $x_i$  to enter the basis: Time  $O(n)$ .

Identify the variable  $x_j$  to leave the basis: Time  $O(m)$ .

Rewrite equation for  $x_j$  to equation for  $x_i$ : Time  $O(n)$ .

Substitute equation for  $x_j$  everywhere: Time  $O(nm)$ .

# Number of improve steps

Number of improve steps

# Number of improve steps

Number of improve steps  
 $\leq$  Number of possible feasible dictionaries

# Number of improve steps

Number of improve steps

$\leq$  Number of possible feasible dictionaries

$\leq$  Number of possible partitions into basic/non-basic variables

# Number of improve steps

$$\begin{aligned} & \text{Number of improve steps} \\ \leq & \text{Number of possible feasible dictionaries} \\ \leq & \text{Number of possible partitions into basic/non-basic variables} \\ = & \binom{m+n}{m} \end{aligned}$$

# Number of improve steps

$$\begin{aligned} & \text{Number of improve steps} \\ \leq & \text{Number of possible feasible dictionaries} \\ \leq & \text{Number of possible partitions into basic/non-basic variables} \\ = & \binom{m+n}{m} \end{aligned}$$

Exponential in  $m$  og  $n$ !

# Exponential time

Assume  $m = n = 50$ .

If we actually have to go through  $\binom{100}{50}$  pivotings on a GigaHertz computer with  $10^9$  pivotings per second, how long time will it take us to terminate?

Assume  $m = n = 50$ .

If we actually have to go through  $\binom{100}{50}$  pivotings on a GigaHertz computer with  $10^9$  pivotings per second, how long time will it take us to terminate?

**4000 billion years!**

# Can we find much better upper bounds?

**Klee-Minty examples:** When we choose the variable with the largest coefficient to enter the basis (largest coefficient rule), we may go through  $2^n - 1$  iterations.

# Can we find much better upper bounds?

**Klee-Minty examples:** When we choose the variable with the largest coefficient to enter the basis (largest coefficient rule), we may go through  $2^n - 1$  iterations.

**Bland rule:** More complicated examples show similar behavior.

# Can we find much better upper bounds?

**Klee-Minty examples:** When we choose the variable with the largest coefficient to enter the basis (largest coefficient rule), we may go through  $2^n - 1$  iterations.

**Bland rule:** More complicated examples show similar behavior.

**Other rules:** More complicated examples show similar behavior.

# Can we find much better upper bounds?

**Klee-Minty examples:** When we choose the variable with the largest coefficient to enter the basis (largest coefficient rule), we may go through  $2^n - 1$  iterations.

**Bland rule:** More complicated examples show similar behavior.

**Other rules:** More complicated examples show similar behavior.

Is there *some* pivot rule that will yield a polynomial upper bound on the number of iterations?

# Can we find much better upper bounds?

**Klee-Minty examples:** When we choose the variable with the largest coefficient to enter the basis (largest coefficient rule), we may go through  $2^n - 1$  iterations.

**Bland rule:** More complicated examples show similar behavior.

**Other rules:** More complicated examples show similar behavior.

Is there *some* pivot rule that will yield a polynomial upper bound on the number of iterations?

**This is an open problem!**

**Countless experimental reports:** For “natural” instances more than  $\approx m \log_2 n$  iterations is rare.

In other words, the simplex algorithm is efficient on most “natural” instances.

**Countless experimental reports:** For “natural” instances more than  $\approx m \log_2 n$  iterations is rare.

In other words, the simplex algorithm is efficient on most “natural” instances.

**Revised simplex algorithm:** The set of basic variables determines the dictionary, so we don't have to store the entire dictionary. This leads to significant speedup for doing each iteration for many sparse instances.

**Chvatal (1983):** *Even though [asymptotic worst case complexity] does reflect to some extent the reasons why practitioners are satisfied by some algorithms and unsatisfied by others, it fails to capture these reasons fully.*

# A Conjecture

**Random pivot rule:** When selecting pivot variables, select a *random* one among the candidates.

# A Conjecture

**Random pivot rule:** When selecting pivot variables, select a *random* one among the candidates.

**QuickSimplex:** Simplex algorithm using the random pivot rule.

**Random pivot rule:** When selecting pivot variables, select a *random* one among the candidates.

**QuickSimplex:** Simplex algorithm using the random pivot rule.

**Conjecture:** For *all* instances, the *expected* number of iterations performed by QuickSimplex is polynomial.

**Random pivot rule:** When selecting pivot variables, select a *random* one among the candidates.

**QuickSimplex:** Simplex algorithm using the random pivot rule.

**Conjecture:** For *all* instances, the *expected* number of iterations performed by QuickSimplex is polynomial.

**This is one of the most important open problems in the theory of Linear Programming.**

# Worst case efficient algorithms strike back

1947. Dantzig - Simplex algorithm - Exponential time but practical.

1979. Khachian - Ellipsoid algorithm - Polynomial time but impractical.

1984. Karmakar - Interior Point algorithm - Polynomial time and practical.

1990's. Interior point algorithms typically beat Simplex algorithm for instance size  $\geq 50000$ .

Mitchell, Pardalos, Resende 1997: *The relative superiority of interior-point methods over the simplex method grows as the problem size grows and the speed-up factor can exceed 1000.*

Maximize  $c^T x$  for  $x \in F = \{x \in (\mathbf{R}^+)^n \mid Ax \leq b\}$

If we **guess**  $x \in F$ , we can say that  $c^T x$  is a **lower bound** for the optimal value without executing the simplex algorithm.

Maximize  $c^T x$  for  $x \in F = \{x \in (\mathbf{R}^+)^n \mid Ax \leq b\}$

If we **guess**  $x \in F$ , we can say that  $c^T x$  is a **lower bound** for the optimal value without executing the simplex algorithm.

Can we make similar easy guesses establishing **upper bounds** for the optimal value?

# Example

**Maximize**  $5x_1 + 6x_2 + 3x_3$

**Subject to**

$$5x_1 + 6x_2 + 3x_3 \leq 50$$

$$4x_1 - 3x_2 + 5x_3 \leq 5$$

$$x_1 + 2x_2 - x_3 \leq 1$$

$$x_1, x_2, x_3 \geq 0$$

# Getting upper bounds

To get an upper bound on the achievable value of any solution, we can look for a positive linear combination of the constraints upper bounding the objective function.

# Getting upper bounds

To get an upper bound on the achievable value of any solution, we can look for a positive linear combination of the constraints upper bounding the objective function.

The best possible upper bound that can be achieved this way can be described by a linear program!

# Weak Duality Theorem

$$A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^{m \times 1}, c \in \mathbf{R}^{n \times 1}.$$

**Primal** program P: Maximize  $c^T x$  under  $Ax \leq b, x \geq 0$ .

**Dual** program D: Minimize  $b^T y$  under  $A^T y \geq c, y \geq 0$ .

*If  $x$  is a feasible solution to P and  $y$  is a feasible solution to D then the value  $c^T x$  is smaller than the value  $b^T y$ .*

# Weak Duality Theorem

$A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^{m \times 1}$ ,  $c \in \mathbf{R}^{n \times 1}$ .

**Primal** program P: Maximize  $c^T x$  under  $Ax \leq b$ ,  $x \geq 0$ .

**Dual** program D: Minimize  $b^T y$  under  $A^T y \geq c$ ,  $y \geq 0$ .

*If  $x$  is a feasible solution to P and  $y$  is a feasible solution to D then the value  $c^T x$  is smaller than the value  $b^T y$ .*

**Proof:**

$$c^T x \leq (y^T A)x \leq y^T (Ax) \leq y^T b$$

If  $P$  is unbounded, then  $D$  is infeasible.

If  $P$  is unbounded, then  $D$  is infeasible.

The dual program to the dual program is the primal program.

If  $P$  is unbounded, then  $D$  is infeasible.

The dual program to the dual program is the primal program.

If  $D$  is unbounded then  $P$  is infeasible.

# (Strong) Duality Theorem

If  $P$  has an optimal solution  $x^*$  then  $D$  has an optimal solution  $y^*$  and  $c^T x^* = b^T y^*$ .

An LP maximization instance can be equivalently expressed as an LP minimization instance **with the same optimal value**.

# A Similar Duality Theorem

Value of **Maximal** flow = Size of **Minimal** cut.

This duality theorem was crucial for showing correctness of Ford-Fulkerson.

The LP duality theorem is similarly tied to the correctness of the simplex algorithm.

# How to find the optimal dual solution

**Maximize**  $5x_1 + 4x_2 + 3x_3$

**Subject to**

$$2x_1 + 3x_2 + x_3 \leq 5$$

$$4x_1 + x_2 + 2x_3 \leq 11$$

$$3x_1 + 4x_2 + 2x_3 \leq 8$$

$$x_1, x_2, x_3 \geq 0$$

Maximize  $z$  subject to  $x_1, x_2, \dots, x_6 \geq 0$  and

$$\begin{aligned}x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\x_5 &= 1 + 5x_2 + 2x_4 \\z &= 13 - 3x_2 - x_4 - x_6\end{aligned}$$

**Opt. Primal solution:**  $x_2 = x_4 = x_6 = 0, x_3 = 1, x_1 = 2, x_5 = 1.$

Maximize  $z$  subject to  $x_1, x_2, \dots, x_6 \geq 0$  and

$$\begin{aligned}x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\x_5 &= 1 + 5x_2 + 2x_4 \\z &= 13 - 3x_2 - x_4 - x_6\end{aligned}$$

**Opt. Primal solution:**  $x_2 = x_4 = x_6 = 0, x_3 = 1, x_1 = 2, x_5 = 1.$

**Opt. Dual solution:**  $y_1 = 1, y_2 = 0, y_3 = 1.$

# Proof of strong duality theorem

**Primal** program P: Maximize  $c^T x$  under  $Ax \leq b$ ,  $x \geq 0$ .

Solve P using two phase simplex method, obtaining optimal solution  $x^*$ .

Last row of last dictionary:

$$z = z^* + \sum_{k=1}^{n+m} \bar{c}_k x_k.$$

Let  $y_i^* = -\bar{c}_{n+i}$ ,  $i = 1, 2, \dots, m$ .

**Must show:**

- 1  $y^*$  is a feasible solution to  
D: Minimize  $b^T y$  under  $A^T y \geq c$ ,  $y \geq 0$ .
- 2  $b^T y^* = z^*$ .

# Proof of strong duality theorem (II)

Last row of last dictionary:

$$z = z^* + \sum_{k=1}^{n+m} \bar{c}_k x_k.$$

Last row of first dictionary

$$z = \sum_{j=1}^n c_j x_j$$

The two expressions are equivalent for all  $x$  in

$$\{x \in \mathbf{R}^{n+m} \mid \forall i \in \{1, \dots, m\} : x_{n+i} = b_i - \sum_{j=1}^n a_{ij} x_j.\}$$

# Proof of strong duality theorem (III)

For all  $x \in \{x \in \mathbf{R}^{n+m} \mid \forall i \in \{1, \dots, m\} : x_{n+i} = b_i - \sum_{j=1}^n a_{ij}x_j\}$ :

$$\begin{aligned}\sum_{j=1}^n c_j x_j &= z^* + \sum_{k=1}^{n+m} \bar{c}_k x_k \\ &= z^* + \sum_{j=1}^n \bar{c}_j x_j + \sum_{i=1}^m (-y_i^*) (b_i - \sum_{j=1}^n a_{ij} x_j) \\ &= (z^* - \sum_{i=1}^m b_i y_i^*) + \sum_{j=1}^n (\bar{c}_j + \sum_{i=1}^m a_{ij} y_i^*) x_j\end{aligned}$$

# Proof of strong duality theorem (IV)

For all  $x \in \mathbf{R}^n$ :

$$(z^* - \sum_{i=1}^m b_i y_i^*) + \sum_{j=1}^n (\bar{c}_j + (\sum_{i=1}^m a_{ij} y_i^*) - c_j) x_j = 0$$

# Proof of strong duality theorem (IV)

For all  $x \in \mathbf{R}^n$ :

$$(z^* - \sum_{i=1}^m b_i y_i^*) + \sum_{j=1}^n (\bar{c}_j + (\sum_{i=1}^m a_{ij} y_i^*) - c_j) x_j = 0$$

First, try to plug in  $x = (0, \dots, 0)$ .

# Proof of strong duality theorem (IV)

For all  $x \in \mathbf{R}^n$ :

$$(z^* - \sum_{i=1}^m b_i y_i^*) + \sum_{j=1}^n (\bar{c}_j + (\sum_{i=1}^m a_{ij} y_i^*) - c_j) x_j = 0$$

First, try to plug in  $x = (0, \dots, 0)$ .

Next, try to plug in  $x = (0, \dots, 0, 1, 0, \dots, 0)$  (position  $j$  has the 1).

# Proof of strong duality theorem (V)

$$\forall j : c_j = \bar{c}_j + \sum_{i=1}^m a_{ij}y_i^*$$

$$\forall j : c_j \leq \sum_{i=1}^m a_{ij}y_i^*$$

so  $y^*$  is a feasible solution to the dual program.

$$z^* = \sum_{i=1}^m b_i y_i^* = b^T y^*$$

so  $y^*$  has the same objective function value as  $x^*$ .

# Consequences of duality theorem (to be seen)

- Software solving LP programs to optimality can be easily **checked** (by running the software on D as well as P).
- Solving linear programs to optimality is as easy as solving systems of linear inequalities (by solving the system  $P, D, c^T x = b^T y.$ )
- Dual simplex algorithm (solve D rather than P) is sometimes faster than primal simplex algorithm.
- Optimal mixed strategies in zero-sum games are “unexploitable” (Von Neuman invented linear programming because of the application to two-player games!).
- Ye’s interior point algorithm works by maintaining a solution to the dual and the primal program simultaneously.
- **In general, one may very often gain tremendous insight into a problem phrased as a linear program by looking at its dual.**

Software solving LP programs to optimality can be easily **checked**.

Give the software the primal program as well as the dual program. The solution to the dual problem is a **certificate** that the solution to the primal program is optimal.

# Linear Inequalities Problem

**Input:**  $A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^m$ .

**Output:** If  $\{x \in \mathbf{R}^n \mid Ax \leq b\} = \emptyset$ , report **Infeasible**, otherwise output  $x$  so that  $Ax \leq b$ .

**Input:**  $A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^m$ ,  $c \in \mathbf{R}^n$ ,

**Output:**  $x \in F$  maximizing  $\langle c, x \rangle$  where  $F = \{x \in \mathbf{R}^n \mid Ax \leq b\}$

$F$  is called the set of **feasible** solutions to the program.

**Exceptions:**

If  $F = \emptyset$ , report **Infeasible**.

If  $\forall v \in \mathbf{R} \exists x \in F : \langle c, x \rangle > v$ , report **Unbounded**.

Convert LP instance to instance  $P$  in standard form.

Check if  $P$  infeasible using LI algorithm. If so, report **Infeasible**.

If not, construct dual  $D$  of  $P$ . Use LI algorithm to find  $x$  and  $y$  so that  $x$  satisfies constraints of  $P$ ,  $y$  satisfies constraints of  $D$  and  $c^T x = b^T y$ .

If no such  $(x, y)$  exist, report **Unbounded**, otherwise return  $x$ .

# The Ellipsoid Method

The ellipsoid algorithm (1979) for Linear Programming works by using the reduction and solving the Linear Inequalities Problem.

The ellipsoid algorithm was the first polynomial time algorithm for Linear Programming but it is unpractical.

# The dual simplex algorithm

Proof of the duality theorem  $\Rightarrow$  Doing the simplex algorithm on the **primal** program can also give us the solution to the **dual** program.

Similarly, doing the simplex algorithm on the **dual** program can also give us the solution to the **primal** program: **The dual simplex algorithm.**

# The dual simplex algorithm

Proof of the duality theorem  $\Rightarrow$  Doing the simplex algorithm on the **primal** program can also give us the solution to the **dual** program.

Similarly, doing the simplex algorithm on the **dual** program can also give us the solution to the **primal** program: **The dual simplex algorithm.**

Can be useful as “empiric” running time of Simplex Algorithm is roughly  $\Theta(m \log n)$ .