DM545

Linear and Integer Programming

Lecture 7
Revised Simplex Method

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Revised Simplex Method

2. Efficiency Issues

Outline

1. Revised Simplex Method

2. Efficiency Issues

Revised Simplex Method

Crucial: pivoting (ie, updating) the tableaux is the most costly part. Several ways to carry out this efficiently, requires matrix description of simplex.

- $\max\{c^T x \mid Ax \leq b, x \geq 0\}$
- $B = \{1 \ldots m\}$
- $N = \{n+1 \ldots n+m\}$
- $A_B = [A_1 \ldots A_m]$
- $A_N = [A_{n+1} \ldots A_{n+m}]$

Standard form

$$\left[\begin{array}{c:c:c:c} A_N & A_B & 0 & b \\ \hdashline c_N & c_B & 1 & 0 \end{array}\right]$$

$$
\begin{aligned}
Ax = A_N x_N + A_B x_B &= b \\
A_B x_B &= b - A_N x_N \\
x_B &= A_B^{-1} b - A_B^{-1} A_N x_N
\end{aligned}
$$

basic feasible solution:

- $X_N = 0$
- A_B lin. indep.
- $X_B \geq 0$

$$
\begin{aligned}
z = cx &= c_B(A_B^{-1} b - A_B^{-1} A_N x_N) + c_N x_N = \\
&= c_B A_B^{-1} b + (c_N - c_B \underbrace{A_B^{-1} A_N}_{\bar{A}}) x_N
\end{aligned}
$$

Canonical form

$$
\left[\begin{array}{c:c:c:c}
A_B^{-1} A_N & I & 0 & A_B^{-1} b \\
\hdashline
c_N^T - C_B^T A_B^{-1} A_N & 0 & 1 & -c_B^T A_B^{-1} b
\end{array}\right]
$$

We do not need to compute all elements of $\bar{A}$

Example

$$\begin{aligned} \max \quad & x_1 + x_2 \\ & -x_1 + x_2 \leq 1 \\ & x_1 \leq 3 \\ & x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & x_1 + x_2 \\ & -x_1 + x_2 + x_3 = 1 \\ & x_1 + x_4 = 3 \\ & x_2 + x_5 = 2 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

x1	x2	x3	x4	x5	-z	b
-1	1	1	0	0	0	1
1	0	0	1	0	0	3
0	1	0	0	1	0	2
1	1	0	0	0	1	0

After two iterations

x1	x2	x3	x4	x5	-z	b
1	0	1	0	-1	0	1
0	1	0	0	-1	0	2
0	0	-1	1	1	0	2
0	0	1	0	-2	1	3

- Basic variables x_1, x_2, x_4. Non basic: x_3, x_5

$$A_B = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad A_N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad x_B = \begin{bmatrix} x_1 \\ x_2 \\ x_4 \end{bmatrix} \quad x_N = \begin{bmatrix} x_3 \\ x_5 \end{bmatrix}$$

$$c_B = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad c_N = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

- **Entering variable**:
  in std. we look at tableau, in revised we need to compute: $c_N - c_B A_B^{-1} A_N$
  1. find $y = c_B A_B^{-1}$ (by solving $yA_B = c_B$, the latter can be done more efficiently)
  2. calculate $c_N - y^T A_N$

Step 1:

$$\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

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

Step 2:

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

(Note that they can be computed individually: $c_j - ya_{ij} > 0$)
Let's take the first we encounter x_3

- **Leaving variable**
  we increase variable by largest feasible amount θ

$$\begin{aligned} \text{I: } x_1 + x_3 - x_5 &= 1 \\ \text{II: } x_2 + 0x_3 - x_5 &= 2 \\ \text{III: } -x_3 + x_4 + x_5 &= 2 \end{aligned} \qquad \begin{aligned} x_1 &= 1 - x_3 \\ &-- \\ x_4 &= 2 + x_3 \end{aligned}$$

$$x_B = x_B^* - A_B^{-1} A_N x_N$$
$$x_B = x_B^* - d\theta$$

d is the column of $A_B^{-1} A_N$ that corresponds to the entering variable, ie, $d = A_B^{-1} a$ where a is the entering column

3. Find θ such that x_B stays positive:
   Find $d = A_B^{-1} a$ (by solving $A_B d = a$)

Step 3:

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \implies d = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \implies x_B = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \theta \geq 0$$

$2 - \theta \geq 0 \implies \theta \leq 2 \rightsquigarrow x_4$ leaves

- So far we have done computations, but now we save the pivoting update. The update of A_B is done by replacing the leaving column by the entering column.

$$x_B^* = \begin{bmatrix} x_1 - d_1\theta \\ x_2 - d_2\theta \\ \theta \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix} \qquad A_B = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- Many implementations depending on how $yA_B = c_B$ and $A_B d = a$ are solved. They are in fact solved from scratch.
- many operations saved especially if many variables!
- special ways to call the matrix A from memory
- better control over numerical issues since A_B^{-1} can be recomputed.

Outline

Solving the two Systems of Equations

$\mathbf{A}_B \mathbf{x} = \mathbf{b}$ solved without computing $\mathbf{A}_B^{-1}$
(costly and likely to introduce numerical inaccuracy)

Recall how the inverse is computed:

For a 2×2 matrix the matrix inverse is

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}^T = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

For a 3×3 matrix the matrix inverse is

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ - \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ + \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}^T$$

Eta Factorization of the Basis

Let $A_B = B$, kth iteration
B_k be the matrix with col p differing from B_{k-1}
Column p is the a column appearing in $B_{k-1}d = a$ solved at 3)
Hence:

$$B_k = B_{k-1}E_k$$

E_k is the eta matrix differing from id. matrix in only one column

$$\begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & & -1 \\ & 1 & 0 \\ & & 1 \end{bmatrix}$$

No matter how we solve $yB_{k-1} = c_B$ and $B_{k-1}d = a$, their update always relays on $B_k = B_{k-1}E_k$ with E_k available.
Plus when initial basis by slack variable $B_0 = I$ and $B_1 = E_1, B_2 = E_1E_2 \cdots$:

$$B_k = E_1E_2 \ldots E_k \qquad \text{eta factorization}$$

$$((((yE_1)E_2)E_3)\cdots)E_k = c_B, \quad uE_4 = c_B,\ vE_3 = u,\ wE_2 = v,\ yE_1 = w$$
$$(E_1(E_2 \cdots E_kd)) = a, \quad E_1u = a,\ E_2v = u,\ E_3w = v,\ E_4d = w$$

LU factorization

Worth to consider also the case of $B_0 \neq I$:

$$B_k = B_0 E_1 E_2 \ldots E_k \qquad \text{eta factorization}$$

$$((((yB_0)E_1)E_2)\cdots)E_k = c_B$$
$$(B_0(E_1 \cdots E_k d)) = a$$

We need an LU factorization of $\mathbf{B}_0$

LU Factorization

To solve the system $\mathbf{Ax} = \mathbf{b}$ by Gaussian Elimination we put the $\mathbf{A}$ matrix in row echelon form by means of elemntary row operations. Each row operation corresponds to multiply left and right side by a lower triangular matrix $\mathbf{L}$ and a permuation matrix $\mathbf{P}$. Hence, the method:

$$\begin{aligned}
\mathbf{Ax} &= \mathbf{b}\\
\mathbf{L}_1\mathbf{P}_1\mathbf{Ax} &= \mathbf{L}_1\mathbf{P}_1\mathbf{b}\\
\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{Ax} &= \mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{b}\\
&\vdots\\
\mathbf{L}_m\mathbf{P}_m\ldots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{Ax} &= \mathbf{L}_m\mathbf{P}_m\ldots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{b}
\end{aligned}$$

thus

$$\mathbf{U} = \mathbf{L}_m\mathbf{P}_m\ldots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{A} \qquad \text{triangular factorization of } \mathbf{A}$$

where $\mathbf{U}$ is an upper triangular matrix whose entries in the diagonal are ones. (if $\mathbf{A}$ is nonsingular such triangularization is unique)

[see numerical example in Va sc 8.1]

We can compute the triangular factorization of $\mathbf{B}_0$ before the initial iterations of the simplex:

$$\mathbf{L}_m \mathbf{P}_m \dots \mathbf{L}_2 \mathbf{P}_2 \mathbf{L}_1 \mathbf{P}_1 \mathbf{B}_0 = \mathbf{U}$$

We can then rewrite $\mathbf{U}$ as

$$\mathbf{U} = \mathbf{U}_m \mathbf{U}_{m-1} \dots, \mathbf{U}_1$$

with each $\mathbf{U}_j$ standing for the eta matrix obtained when the jth column of $\mathbf{I}$ is replaced by the jth column of $\mathbf{U}$. Hence:

$$\mathbf{L}_m \mathbf{P}_m \dots \mathbf{L}_2 \mathbf{P}_2 \mathbf{L}_1 \mathbf{P}_1 \mathbf{B}_k = \mathbf{U}_m \mathbf{U}_{m-1} \dots \mathbf{U}_1$$

Then $\mathbf{y}\mathbf{B}_k = \mathbf{c}_B$ can be solved by first solving:

$$((((\mathbf{y}\mathbf{U}_m)\mathbf{U}_{m-1}) \cdots)\mathbf{E}_k = \mathbf{c}_B$$

and then replacing $\mathbf{y}$ by $(\mathbf{y}\mathbf{L}_m\mathbf{P}_m) \cdots)\mathbf{L}_1\mathbf{P}_1$.
E_i matrices can be stored by only storing the column and the position. If sparse columns then can be stored in compact mode, ie only nonzero values and their indices. Same for the triangular eta matrices $\mathbf{L}_j$, $\mathbf{U}_j$ while for $\mathbf{P}_j$ just two indices are needed.

- Solving $\mathbf{y}\mathbf{B}_k = \mathbf{c}_B$ also called backward transformation (BTRAN)
- Solving $\mathbf{B}_k\mathbf{d} = \mathbf{a}$ also called forward transformation (FTRAN)

More on LP

- Tableau method is unstable: computational errors may accumulate. Revised method has a natural control mechanism: we can recompute A_B-1 at any time
- Commercial and freeware solvers differ from the way the systems $\mathbf{y} = \mathbf{c}_B\mathbf{A}_B^{-1}$ and $\mathbf{A}_B\mathbf{d} = \mathbf{a}$ are resolved

Efficient Implementations

- Dual simplex with steepest descent
- Linear Algebra:
  - Dynamic LU-factorization using Markowitz threshold pivoting (Suhl and Suhl, 1990)
  - sparse linear systems: Typically these systems take as input a vector with a very small number of nonzero entries and output a vector with only a few additional nonzeros.
- Presolve, ie problem reductions: removal of redundant constraints, fixed variables, and other extraneous model elements.
- dealing with degeneracy, stalling (long sequences of degenerate pivots), and cycling:
  - bound-shifting (Paula Harris, 1974)
  - Hybrid Pricing (variable selection): start with partial pricing, then switch to devex (approximate steepest-edge, Harris, 1974)
- A model that might have taken a year to solve 10 years ago can now solve in less than 30 seconds (Bixby, 2002).

Further topics in LP

- Ellipsoid method: cannot compete in practice but polynomial time (Khachyian, 1979)
- Interior point algorithm(s) (Karmarkar, 1984) competitive with simplex and polynomial in some versions
  - iterate through points interior to the feasibility region
  - because of patents reasons, also known as barrier algorithm
  - one single iteration is computationally more intensive than the simplex
  - particularly competitive in presence of many constraints (eg, for $m = 10,000$ may need less than 100 iterations)
  - bad for post-optimality analysis $\rightsquigarrow$ crossover algorithm to convert a sol of barrier method into a basic feasible solutions for the simplex
- Lagrangian relaxation
- Column generation
- Decomposition methods:
  - Dantzig Wolfe decomposition
  - Benders decomposition

Interior Point Algorithm

1. Start at an interior point of the feasible region
2. Move in a direction that improves the objective function value at the fastest possible rate
3. Transform the feasible region to place the current point at the center of it

How Large Problems Can We Solve?

Very large model

	Rows	Columns	Nonzeros
Original size	5034171	7365337	25596099
After presolve	1296075	2910559	10339042

Solution times were as follows:

Very large model—solution times

	Algorithm		
Version	Barrier	Dual	Primal
CPLEX 5.0	8642.6	350000.0	71039.7
CPLEX 7.1	5642.6	6413.1	1880.0

Source: Bixby, 2002