ORF 522: Lecture 4
Linear Programming: Chapter 4
Efficiency

Robert J. Vanderbei

September 25, 2012

Slides last edited on September 27, 2012
Efficiency

Question:

Given a problem of a certain size, how long will it take to solve it?

Two Kinds of Answers:

- **Average Case.** How long for a *typical* problem.
- **Worst Case.** How long for the *hardest* problem.

**Average Case.**

- Mathematically difficult.
- Empirical studies.

**Worst Case.**

- Mathematically tractible.
- Limited value.
Measures

**Measures of Size**

- Number of constraints $m$ and/or number of variables $n$.
- Number of data elements, $mn$.
- Number of nonzero data elements.
- Size, in bytes, of AMPL formulation (model+data).

**Measuring Time**

Three factors:
- Number of iterations.
- Arithmetic operations per iteration.
- Time per arithmetic operation (depends on hardware).
Klee–Minty Problem (1972)

maximize \[ \sum_{j=1}^{n} 2^{n-j} x_j \]
subject to \[ 2 \sum_{j=1}^{i-1} 2^{i-j} x_j + x_i \leq 100^{i-1} \quad i = 1, 2, \ldots, n \]
\[ x_j \geq 0 \quad j = 1, 2, \ldots, n. \]

Example \( n = 3 \):

maximize \[ 4x_1 + 2x_2 + x_3 \]
subj. to \[ x_1 \leq 1 \]
\[ 4x_1 + x_2 \leq 100 \]
\[ 8x_1 + 4x_2 + x_3 \leq 10000 \]
\[ x_1, x_2, x_3 \geq 0. \]
A Distorted Cube

Case $n = 3$:

Constraints represent a “minor” distortion to an $n$-dimensional hyper-cube:

\[ 0 \leq x_1 \leq 1 \]
\[ 0 \leq x_2 \leq 100 \]
\[ \vdots \]
\[ 0 \leq x_n \leq 100^{n-1}. \]
Analysis

Replace

\[ 1, 100, 10000, \ldots, \]

with

\[ 1 = b_1 \ll b_2 \ll b_3 \ll \ldots. \]

Then, make following replacements to rhs:

\[
\begin{align*}
b_1 & \rightarrow b_1 \\
b_2 & \rightarrow 2b_1 + b_2 \\
b_3 & \rightarrow 4b_1 + 2b_2 + b_3 \\
b_4 & \rightarrow 8b_1 + 4b_2 + 2b_3 + b_4 \\
\vdots
\end{align*}
\]

Hardly a change!

Make a similar constant adjustment to objective function.

Look at the pivot tool version...
Case $n = 3$:

<table>
<thead>
<tr>
<th></th>
<th>obj</th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>w1</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w2</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>w3</td>
<td>-1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Now, watch the pivots...
Klee–Minty problem shows that:

Largest-coefficient rule can take $2^n - 1$ pivots to solve a problem in $n$ variables and constraints (thereby visiting all $2^n$ vertices of the distorted cube).

For $n = 70$,

$$2^n = 1.2 \times 10^{21}.$$  

At 1000 iterations per second, this problem will take 40 billion years to solve. The age of the universe is estimated at 13.7 billion years.

Yet, problems with 10,000 to 100,000 variables are solved routinely every day.

Worst case analysis is just that: worst case.
### Complexity

<table>
<thead>
<tr>
<th>n</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>216</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>343</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>729</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>1024</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>1728</td>
<td>4096</td>
</tr>
<tr>
<td>14</td>
<td>196</td>
<td>2744</td>
<td>16384</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>4096</td>
<td>65536</td>
</tr>
<tr>
<td>18</td>
<td>324</td>
<td>5832</td>
<td>262144</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>8000</td>
<td>1048576</td>
</tr>
<tr>
<td>22</td>
<td>484</td>
<td>10648</td>
<td>4194304</td>
</tr>
<tr>
<td>24</td>
<td>576</td>
<td>13824</td>
<td>16777216</td>
</tr>
<tr>
<td>26</td>
<td>676</td>
<td>17576</td>
<td>67108864</td>
</tr>
<tr>
<td>28</td>
<td>784</td>
<td>21952</td>
<td>268435456</td>
</tr>
<tr>
<td>30</td>
<td>900</td>
<td>27000</td>
<td>1073741824</td>
</tr>
</tbody>
</table>

**Sorting:** $= n \log n$

**Matrix times vector:** $n^2$

**Matrix times matrix:** $n^3$

**Matrix inversion:** $n^3$

**Simplex Method:**
- **Worst case:** $n^2 2^n$ operations.
- **Average case:** $n^3$ operations.
- **Open question:** Does there exist a variant of the simplex method whose worst case performance is polynomial?

**Linear Programming:**
- **Theorem:** There exists an algorithm whose worst case performance is $n^{3.5}$ operations.
Define a random problem:

\[
m = \text{ceil}(\exp(\log(400) \times \text{rand}()));
\]
\[
n = \text{ceil}(\exp(\log(400) \times \text{rand}()));
\]

\[
A = \text{round}(\sigma \times \text{randn}(m,n));
\]
\[
b = \text{round}(\sigma \times \text{rand}(m,1));
\]

\[
y = \text{round}(\sigma \times \text{rand}(1,m));
\]
\[
z = \text{round}(\sigma \times \text{rand}(1,n));
\]
\[
c = y \times A - z;
\]

\[
A = -A;
\]

Initialize a few things:

\[
\text{iter} = 0;
\]
\[
\text{opt} = 0;
\]
The Main Loop:

while max(c) > eps,
    % pick largest coefficient
    [cj, col] = max(c);
    Acol = A(:,col);

    % select leaving variable
    [t, row] = max(-Acol./b);
    if t < eps,
        opt = -1; % unbounded
        'unbounded'
        break;
    end
    Arow = A(row,:);
    a = A(row,col); % pivot element

    .
    .
    .
    iter = iter+1;
end

The code for a pivot:

A = A - Acol*Arow/a;
A(row,:) = -Arow/a;
A(:,col) = Acol/a;
A(row,col) = 1/a;

brow = b(row);
b = b - Acol*brow/a;
b(row) = -brow/a;

ccol = c(col);
c = c - ccol*Arow/a;
c(col) = ccol/a;
Primal Simplex Method

$m + n$

Number of pivots

$m + n$ vs. number of pivots
Primal Simplex Method

minimum of $m$ and $n$

number of pivots

\[ \text{iters} = 0.6893 \min(m, n)^{1.3347} \approx \frac{2}{3} \min(m, n)^{4/3} \]
Primal Simplex Method

The number of pivots is plotted against the minimum of m and n.
Declare parameters:

```plaintext
param eps := 1e-9;
param sigma := 30;
param niters := 1000;
param size := 400;

param m;
param n;
param AA {1..size, 1..size};
param bb {1..size};
param cc {1..size};
param A {1..size, 1..size};
param b {1..size};
param c {1..size};
param x {1..size};
param y {1..size};
param w {1..size};

param iter;
param opt;
param stop;
param mniters {1..niters, 1..3};
param maxc;
param minbovera;
param col;
param row;
```

Define a random problem:

```plaintext
let m := ceil(exp(log(size)*Uniform01()));
let n := ceil(exp(log(size)*Uniform01()));
let {i in 1..m, j in 1..n} A[i,j] := round(sigma*Normal01());
let {i in 1..m} y[i] := round(sigma*Uniform01());
let {j in 1..n} z[j] := round(sigma*Uniform01());
let {i in 1..m} b[i] := round(sigma*Uniform01());
let {j in 1..n} c[j] := sum {i in 1..m} y[i]*A[i,j] - z[j];
let {i in 1..m, j in 1..n} A[i,j] := -A[i,j];
let {i in 1..m, j in 1..n} AA[i,j] := A[i,j];
let {i in 1..m} bb[i] := b[i];
let {j in 1..n} cc[j] := c[j];
```
The Simplex Method (Phase 2)

repeat while (max {j in 1..n} c[j]) > eps {
    let maxc := 0;
    for {j in 1..n} {
        if (c[j] > maxc) then {
            let maxc := c[j];
            let col := j;
        }
    }
    let minbovera := 1/eps;
    for {i in 1..m} {
        if (A[i,col] < -eps) then {
            if (-b[i]/A[i,col] < minbovera) then {
                let minbovera := -b[i]/A[i,col];
                let row := i;
            }
        }
    }
    if minbovera >= 1/eps then {
        let opt := -1; # unbounded
        display "unbounded";
        break;
    }
    .
    .
    .
}

The code for a pivot:

let {j in 1..n} Arow[j] := A[row,j];
let {i in 1..m} Acol[i] := A[i,col];
let a := A[row,col];
let {i in 1..m, j in 1..n}
let {j in 1..n} A[row,j] := -Arow[j]/a;
let {i in 1..m} A[i,col] := Acol[i]/a;
let A[row,col] := 1/a;

let brow := b[row];
let {i in 1..m}
    b[i] := b[i] - brow*Acol[i]/a;
let b[row] := -brow/a;

let ccol := c[col];
let {j in 1..n}
    c[j] := c[j] - ccol*Arow[j]/a;
let c[col] := ccol/a;
The AMPL code can be found here:

http://orfe.princeton.edu/ rvdb/307/lectures/primalsimplex.txt