**Linear Optimization Spring 2016 Project**

1. Problem Formulation

max 

s.t.







Where

n = total product( i )

m = machines available( j )

p = total months( k )

1. Strategy used to generate the LP instances
   1. Created example A, h, d, c, and x matrices based on the above LP and the parameters n = 2, m = 2, and p = 3.
   2. Based on the above LP and the example matrix, constructed from the previous matrices, we broke the problem down into logical steps and developed pseudo code based on those steps.
   3. Incrementally developed code to achieve the desired results of the pseudo code and logic.
   4. Tested and debugged the code until it was ready for implementation.
   5. Implemented the code with a variety of parameters to acquire a wide range of results.

2.1 Pseudo Code for GLPK function:

I. Generate [n, m, p] values // We choose these values

II. Build Matrix A with appropriate dimensions

a. Set every element in A to (0)

- Columns = (m \* n \* p)

- Rows = (n \* p) + (m \* p)

III. Build Matrix C

- Rows = 1

- Columns = Same as A

IV. Build Matrix B

- Rows = Same as A

- Columns = 1

V. Generate all required parameter structures with appropriate members

a. xijk structures: required (n \* m \* p) instances

Members:

- (i) instance of total products

- (j) instance of machines available

- (k) instance of total months

- Columns // Used for indexing the A matrix

b. hjk structures: required ( m \* p) instances

Members:

- (j) instance of machines available

- (k) instance of total months

- mHours // Machine Hours

- row // Used for indexing the A matrix

c. dik structures required = (n \* p)

Members:

- (i) instance of total products

- (k) instance of total months

- demand // Demand of the product

- row2 // Used for indexing the A matrix after the h’s are done

d. cik structures required = (n \* m \* p)

Members:

- (i) instance of total products

- (k) instance of total months

- cost // cost of the product

e. tij structures: required (n \* m \* p) instances

Members:

- (i) instance of total products

- (j) instance of machines available

- time // How much time it takes to create an item

VI. Set the appropriate times for the elements of A corresponding with the indices of X.

VII. Flag the appropriate demands for the products in matrix A.

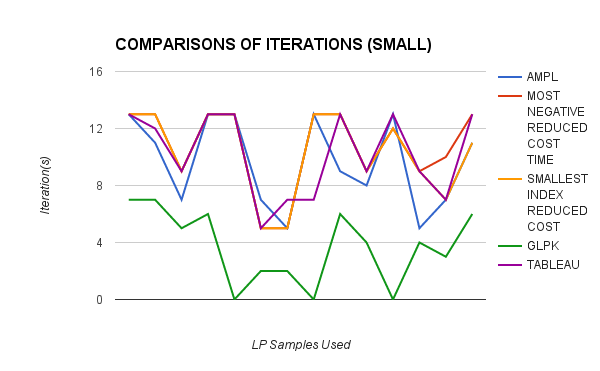
VIII. Example code implementation:

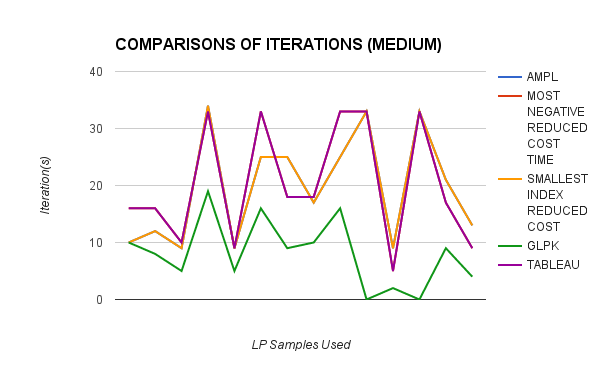
--------------------------------------------------------------------------------------------------------------------------------

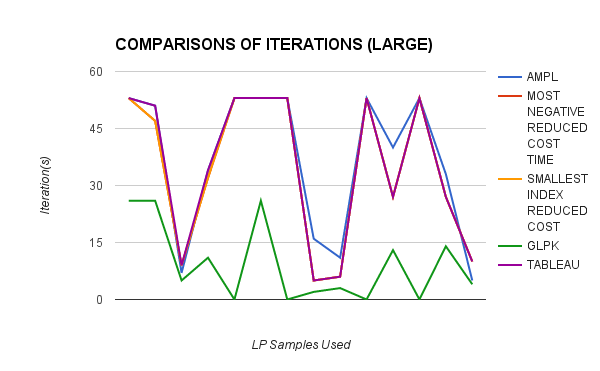
n = 2 m = 2 p = 3

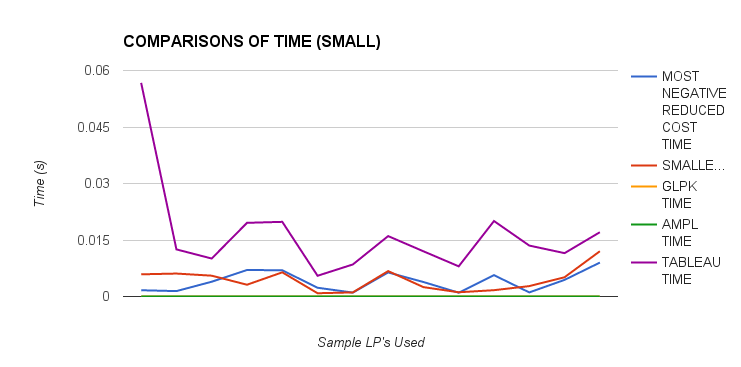
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x.ijk** | x.111 | x.112 | x.113 | x.121 | x.122 | x.123 | x.211 | x.212 | x.213 | x.221 | x.222 | x.223 | **h.jk** |
| **t.ij** | t.11 | 0 | 0 | 0 | 0 | 0 | t.21 | 0 | 0 | 0 | 0 | 0 | h.11 |
|  | 0 | t.11 | 0 | 0 | 0 | 0 | 0 | t.21 | 0 | 0 | 0 | 0 | h.12 |
|  | 0 | 0 | t.11 | 0 | 0 | 0 | 0 | 0 | t.21 | 0 | 0 | 0 | h.13 |
|  | 0 | 0 | 0 | t.12 | 0 | 0 | 0 | 0 | 0 | t.22 | 0 | 0 | h.21 |
|  | 0 | 0 | 0 | 0 | t.12 | 0 | 0 | 0 | 0 | 0 | t.22 | 0 | h.22 |
|  | 0 | 0 | 0 | 0 | 0 | t.12 | 0 | 0 | 0 | 0 | 0 | t.22 | h.23 |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **d.ik** |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d.11 |
|  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d.12 |
|  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | d.13 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | d.21 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | d.22 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | d.23 |
| c.ik | c.11 | c.12 | c.13 | c.11 | c.12 | c.13 | c.21 | c.22 | c.23 | c.21 | c.22 | c.23 |  |

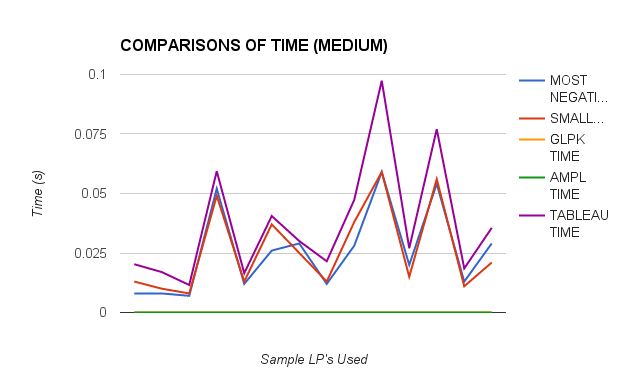
1. Details and comparisons of running times and iterations
   1. Three LP sizes were generated to be tested in AMPL and Octave:
      1. Small (n\*m\*p = 12) 1-14 Samples
      2. Medium (n\*m\*p = 32) 15-28 Samples
      3. Large (n\*m\*p = 52) 29-42 Samples
   2. The total number of generated LP’s was 42.
   3. The LP’s were then tested in
      1. Revised Simplex Method function (Most negative reduced cost variable leaving AND smallest index reduced cost variable leaving were tested. Smallest index variable leaving)
      2. Tableau Method function (Most negative reduced cost entering variable and smallest index variable leaving)
      3. AMPL
      4. GLPK using Octave
   4. The run times and number of iterations of each run for each method were saved and then put into six separate graphs: small, medium, and large for iterations and again for the time comparisons.
   5. Below are the graphs of the comparisons.

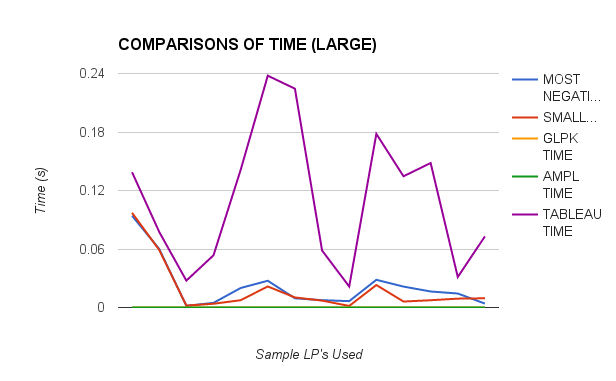












1. Time Comparisons:
   1. AMPL and GLPK both took the smallest amount of time to compute LP’s, with numbers so low that they were generally rounded to zero.
   2. The slowest function was the Tableau Method which I implemented using Octave.
   3. The Revised Simplex Method took similar times for both entering variable options.
2. Iteration Comparisons:
   1. GLPK was very efficient in the number of iterations used to solve an LP. The GLPK function must be extremely optimized to be able to solve in such a low number of iterations.
   2. The rest of the functions (AMPL, Simplex Method, and Tableau Method) all took similar amounts of iterations to solve the LP’s. Though it seems in the Simplex Method combined with *smallest index* negative reduced cost entering variable on average took fewer iterations than using *most* negative reduced cost entering variable.