Multiple Knapsack Problem

BANA 6800 - Project Paper

By
Brian Cleary
Brandon Crain
Avinash Kamath
Pavani Korada

Introduction

The knapsack problem is a problem in combinatorial optimization. The name "knapsack problem" has been referenced in the early works of mathematician Tobias Dantzig [1]. The problem states that, given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
Mathematically, the Knapsack problem can be written as follows:

$$\text{maximize} \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} w_j x_j \leq W,$$

$$x_j \in \{0,1\} \qquad \forall j \in \{1,\ldots,n\}$$

Where
n = set of items ($1<j<n$)
p_j = profit of item j
w_j = weight of item j
W = capacity of knapsack
x_j = binary decision variables to select an item

Here the objective is to pick the items to maximize the total profit while ensuring that the total weight of the chosen items do not exceed the maximum weight of the knapsack.
Some variants of the Knapsack problem are:

- The **Bounded Knapsack problem** specifies, for each item *j*, an upper bound u_j (which may be a positive integer, or infinity) on the number of times item *j* can be selected.
- The **Unbounded knapsack problem** does not put any upper bounds on the number of times an item j may be selected.
- The **Multiple- Choice Knapsack Problem** applies if the items are subdivided into *k* classes denoted N_i, and exactly one item must be taken from each class.
- The **Multiple Knapsack Problem** applies if we have m knapsacks each with a capacity Wi, that the n items can be placed into.

Multiple Knapsack Problem

Mathematically, the problem statement can be written as

$$\text{maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$\text{subject to} \sum_{j=1}^{n} w_j x_{ij} \leq W_i, \text{ for all } 1 \leq i \leq m$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \qquad \text{for all } 1 \leq j \leq n$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } 1 \leq j \leq n \text{ and all } 1 \leq i \leq m$$

For the purpose of this report, we assumed that w_j, p_j, W_j are positive numbers.

Applications

This multiple knapsack problem has applications in decision making processes such as finding least wasteful ways to cut raw materials, selection of investments and portfolios[2], project selections, assignment of files to storage devices in order to maximize the number of files stored in the fastest storage devices, transportation, cargo airline dispatching[3] and cryptography[4]

Computational Analysis

According to Petersen [5], companies engaged in contract procurement can optimize contract volume with a binary, integer programing. The contract volume is measured in dollars and represents the sum of benefits realized. Optimizing contract volume ensures optimal benefit is realized, while managing scarce resources. This matters because successful bidders often spend several years working on pre-contract work—time is money. As is often the case with knapsack problems, any "must-have" projects are included in the selection process because they will be selected regardless of other selections.

For this report, we used the test problems referenced in the Petersen [5] paper and modelled them in AMPL[6]. The paper reviews the use of the multiple knapsack problem to aid the selection of projects that will maximize the contract dollar volume while staying within the budget constraints.

All variables have a non-negativity constraint and are as follows:

Objective Function (maximize): $\sum c_j x_j$, subject to $\sum\sum a_{ij} x_j$
N = potential projects to select from
M = time periods
zmax = maximum profit from the selected projects
x_j = Binary variable for project selection (1 if chosen, 0 if not chosen)
c_j = Profit estimate for the j^{th} project (j = 1, 2, … N)
a_{ij} = Cost of each project for the i^{th} time period (i = 1, 2,…M)
b_i = resource limit for the i^{th} time period (i = 1, 2,…M)

As part of this report, we compare results and performances AMPL's CPLEX with Gurobi optimizers. CPLEX optimizer has been around longer and is more widely used. Gurobi often performs at least as good as CPLEX in standard benchmarks—an on occastion better than CPLEX.

Metrics used to evaluate the speed* of each solver are as follows:

System time: Total time taken to just load the problem on the machine
User time: Total time taken to just solve the problem
Solve time: Total time taken to load and solve the problem. This is also equal to System time + User time

* Solve times differ as a result of the number of iterations, the method (algorithm) that the optimizer chooses to employ and number of nodes and the type of algorithm that are employed to optimize (for integer problems)

Iterations:
Iterations are the number of loops that the optimizer takes to identify the optimum solution. Higher the number of iterations, more the time required by the optimizer to optimize the solution. These iterations are dependent on the method that the solver chooses to optimize.

Optimizing Methods (Algorithms):
Each optimizer has a fairly complicated internal algorithm for selecting which method it should use to solve linear programs. There are several methods to choose from that include simplex method, dual-simplex method, interior point method, convex or non-convex quadratic problem method etc. In our test problems, we noticed that CPLEX employed Mixed Integer Programming (MIP) simplex method and Gurobi employed simplex method.

Branch and bound nodes vs Branch and cut nodes for integer programming:
B&B is an algorithm written to solve mixed integer linear programming problems by creating branches. It starts first with relaxing the integer constraint and finding the optimum solution. Then it branches out at the variable that is fractional and identifies solutions at the upper and lower bounds. The process is repeated at the upper and lower integer bounds of the branched variable. This leads to formation of the branches of a tree that represent the subset of the possible solutions. At each node, the algorithm identifies the upper and lower bounds and discards the whole set if it cannot identify a better solution compared to the best one that already exists.

B&C is an algorithm that speeds up the B&B algorithm by using a divide and conquer approach. This is done by cutting planes at the top of a B&B of a tree or at every node of a tree because cutting planes cutting planes considerably reduce the size of a tree. Branch & bound = Branch & cut + Cutting planes

Test Problem 1: Here we have **28 projects** to choose from, that span the course of ten years. The project cost for each year along with the budgets for each year are also specified. A snippet of the data set is provided below:

j	$\mathbf{c_j}$	$\mathbf{a_{1j}}$	$\mathbf{a_{2j}}$	$\mathbf{a_{3j}}$	$\mathbf{a_{4j}}$	$\mathbf{a_{5j}}$	$\mathbf{a_{6j}}$	$\mathbf{a_{7j}}$	$\mathbf{a_{8j}}$	$\mathbf{a_{9j}}$	$\mathbf{a_{10j}}$
1	100	8	8	3	5	5	5	0	3	3	3
2	220	24	44	6	9	11	11	0	4	6	8
…	…	…	…	…	…	…	…	…	…	…	…
28	520	18	28	10	20	20	20	10	20	28	28
$\mathbf{b_i}$		930	1210	272	462	532	572	240	400	470	490

The AMPL model and data file are included in the Appendix 1. The optimal solutions using Cplex and Gurobi solver are provided below

	Cplex Solver	Gurobi Solver
Optimal	zmax =12400	zmax =12400

solution	xj = 1 (j= 1,2,3,9,14,15,16,17,18,19, 20, 21, 22, 23, 25,26,27,28); all others =0	xj = 1 (j= 1,2,3,9,14,15,16,17,18, 19,20,21,22,23, 25,26,27,28); all others =0
Solver Details	CPLEX 12.6.3.0: optimal integer solution; objective 12400 17 MIP simplex iterations 0 branch-and-bound nodes No basis.	Gurobi 6.5.0: optimal solution; objective 12400 56 simplex iterations 1 branch-and-cut nodes
Solve times	solvetime = 0.296402 systemtime = 0.0468003 usertime = 0.249602	solvetime = 0.0624004 systemtime = 0.0156001 usertime = 0.0468003

Test Problem 2: The dataset from above is used, with additional constraints. Some projects are specified to be mutually exclusive while some depend on other projects. Below is a table of all project dependencies. For example Project 4 and 5 and mutually exclusive while projects 4,5,6,7 depend on project 3.

j	Not With	Depends On
1		
2		
3		
4	5	3
5	4	3
6		3
7		3
8	11	
9	14	
10		
11	8	2
12		
13		
14	9	

j	Not With	Depends On
15		
16		
17	18	
18	17	
19		
20		19
21		
22	23	
23	22	
24		23
25		
26	28	
27		
28	26	

The AMPL model and data file are included in the Appendix 2. The optimal solutions using Cplex and Gurobi solver are provided below

	Cplex Solver	Gurobi Solver
Optimal solution	zmax =11690 xj = 1 (j= 1,2,3,8,10,14,15,16,17,19, 20, 21, 22, 25,27,28); all others =0	zmax =11690 xj = 1 (j= 1,2,3,8,10,14,15,16,17,19, 20, 21, 22, 25,27,28); all others =0
Solver Details	CPLEX 12.6.3.0: optimal integer solution; objective 11690 7 MIP simplex iterations 0 branch-and-bound nodes No basis.	Gurobi 6.5.0: optimal solution; objective 11690 11 simplex iterations

Solve times	solvetime = 0.0780005 systemtime = 0.0312002 usertime = 0.0468003	solvetime = 0.0156001 systemtime = 0.0156001 usertime = 0

Test Problem 3: Here we have **50 projects** to choose from, that span the course of 5 years. The project cost for each year along with the budgets for each year are also specified. A snippet of the data set is provided below:

j	$\mathbf{c_j}$	$\mathbf{a_{1j}}$	$\mathbf{a_{2j}}$	$\mathbf{a_{3j}}$	$\mathbf{a_{4j}}$	$\mathbf{a_{5j}}$
1	560	40	16	38	8	38
2	1125	91	92	39	71	52
…	…	…	…	…	…	…
50	81	4	2	2	1	0
$\mathbf{b_i}$		800	650	550	550	600

The AMPL model and data file are included in the Appendix 3. The optimal solutions using Cplex and Gurobi solver are provided below

	Cplex Solver	Gurobi Solver
Optimal solution	zmax =16,537 x_j = 1 (j= 6, 11, 26, 1, 36, 41,12, 17, 27, 32, 37, 42, 47, 8, 13, 23, 28, 38, 43, 48, 4, 9, 19, 29, 34, 39, 44, 49, 15, 20, 25, 35, 40, 50) all others = 0	zmax =16,537 xj = 1 (j= 6, 11, 16, 26, 31, 36, 41, 12, 17, 27,32, 37, 42, 47, 8, 13, 23, 28, 38, 43, 48, 4, 9, 19, 29, 34, 39, 44, 49, 15, 20, 25, 35, 40, 50) all others = 0
Solver Details	CPLEX 12.6.3.0: optimal integer solution; objective 16537 52 MIP simplex iterations 0 branch-and-bound nodes No basis.	Gurobi 6.5.0: optimal solution; objective 16537 186 simplex iterations 50 branch-and-cut nodes
Solve times	Solve time = 0. 0.085577 System time = 0.022964 User time = 0.062613	Solve time = 0.068162 System time = 0.012005 User time = 0.056157

Test Problem 4: The dataset from above is used, with additional modifications and constraints. Instead of strict adherence to a 5-year budget for one company, an R&D and Marketing department were created for a 3-year period. Departments were allowed to roll-over any money not spent in year one in year two—allowing managers to optimize the entire budget rather than just annual. This was accomplished by assigning variables a_{1j} and a_{2j} to R&D (year 1 and 2), while a_{3j}, a_{4j}, and a_{5j} represent Marketing (year 1, 2, and 3). Carry over of was accomplished by summing creating a rolling total for each department by year as shown below.

		R&D		Marketing		
j	$\mathbf{c_j}$	$\mathbf{a_{1j}}$	$\mathbf{a_{2j}}$	$\mathbf{a_{3j}}$	$\mathbf{a_{4j}}$	$\mathbf{a_{5j}}$
1	560	40	56	38	46	84
2	1125	91	183	39	110	162
…	…	…	…	…	…	…
50	81	4	6	2	3	3

$\mathbf{b_i}$		800	1450	550	1100	1750

The AMPL model and data file are included in the Appendix 4. The optimal solutions using Cplex and Gurobi solver are provided below

	Cplex Solver	Gurobi Solver
Optimal solution	zmax = 16,774 $x_j = 1$ (j= 1, 6, 11, 16, 26, 31, 41, 7, 17, 22, 32, 37, 42, 47, 8, 13, 23, 38, 43, 48, 4, 14, 19, 24, 29, 39, 44, 49, 15, 20, 25, 30, 35, 40, 50) All others = 0	zmax = 16,774 $x_j = 1$ (j= 1, 6,11, 16, 26, 31, 41, 7, 17, 22, 32, 37, 42, 47, 8, 13, 23, 38, 43, 48 4, 14, 19, 24, 29, 39, 44, 49, 15, 20, 25, 30, 35, 40, 50) All others =0
Solver Details	CPLEX 12.6.3.0: optimal integer solution Objective 16774 60 MIP simplex iterations 0 branch-and-bound nodes No basis.	Gurobi 6.5.0: optimal solution Objective 16774 172 simplex iterations 61 branch-and-cut nodes
Solve times	Solve time = 0. 0.067098 System time = 0. 0.011783 User time = 0. 0.055315	Solve time = 0.053335 System time = 0.005762 User time = 0.047573

Solver Performance Testing

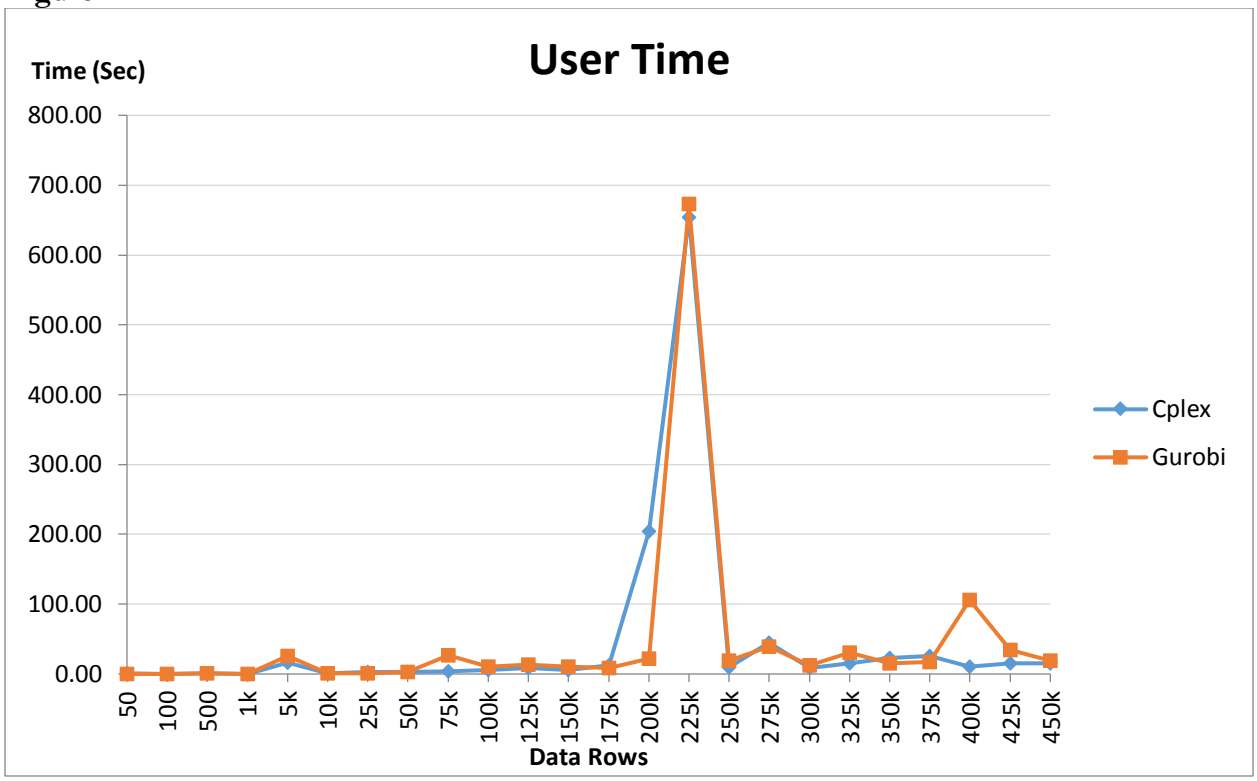
Two popular solvers available in AMPL to solve integer or binary models are CPLEX and Gurobi. Each solver uses a proprietary algorithm to find an optimal solution. The exact way in which each solver calculates the optimal solution is unknown to the user, but recording calculation times for each can give us an idea of what's going on behind the scenes. The multiple knapsack problem uses binary decision variables to select a set of items by multiplying either a 0 or 1 to a value associated with each item. As previously mentioned, a common application for the multiple knapsack problem is in selecting a set of projects to maximize profit or revenue while subject to project cost constraints. In practice, these applications typically have a fairly small number of projects to choose from as it is often easy to weed out the majority of projects based on some preliminary criteria. From the test problems in the previous section, we've learned that both solvers are extremely efficient and take almost no time at all to calculate an optimal solution. To analyze solver performance, we needed to work with a larger dataset which would allow us to record measurable performance metrics.

Using data collected from lendingclub.com[7], we constructed a model that would select a set of available loans in which to invest. We viewed the data from the point of view of a banker who would like to select a set of borrowers for which to lend money so as to maximize expected profit received in the form of interest, subject to available funds and a maximum level of risk the banker is willing to accept in each time period. The model assumes that if a loan is selected, the full amount of the principle is delivered to the borrower. To calculate a value of risk, we used an average rate of loan default associated with each potential borrower's credit grade. The principal is then multiplied by this rate to give us a monetary value of risk for each loan in year one. Risk values for subsequent years are calculated by multiplying the remaining unpaid principal by the same average rate of default until the loan is paid in full. See Appendix 6 for a subset of this data.

Setting up the data in this way allowed us to use a slightly modified version of the model we built to select from a small list of projects, but were able to use a data set with nearly 500,000 loans for the model to select from. We started with a small number of loans in the data set (50), (See Appendix 5 for this model file), solved the model with each of the solvers mentioned above, then recorded the solver time, system time, and user time that each took to find the optimal solution. We then repeated this process while gradually adding additional rows to the data set.

The results of this test were surprising. While the calculation times did generally increase as more rows were added to the data set, they did not increase by as much as we expected. The times also seemed to fluctuate wildly depending on the size of the data set, particularly with the set containing 225k rows. The calculation times for both solvers increased dramatically with this particular dataset, but then decreased by just as much with even larger data sets. Figures 1 - 3 show these results plotted on line charts. To see the upward trend more clearly, we also plotted the results without the 255k row data set since the calculation was an outlier. These results can be seen in figures 4 - 6.

Figure 1



The peaks in calculation times appear to cyclical which may tell us something about the way in which these solvers are finding the optimal solutions. While CPLEX shared the same spike in calculation times as Gurobi with the 225k row data set, Gurobi appears to have a spike about every 75k rows. This likely has something to do with the number of variables the solver is able to rule out with presolving.

Figure 2

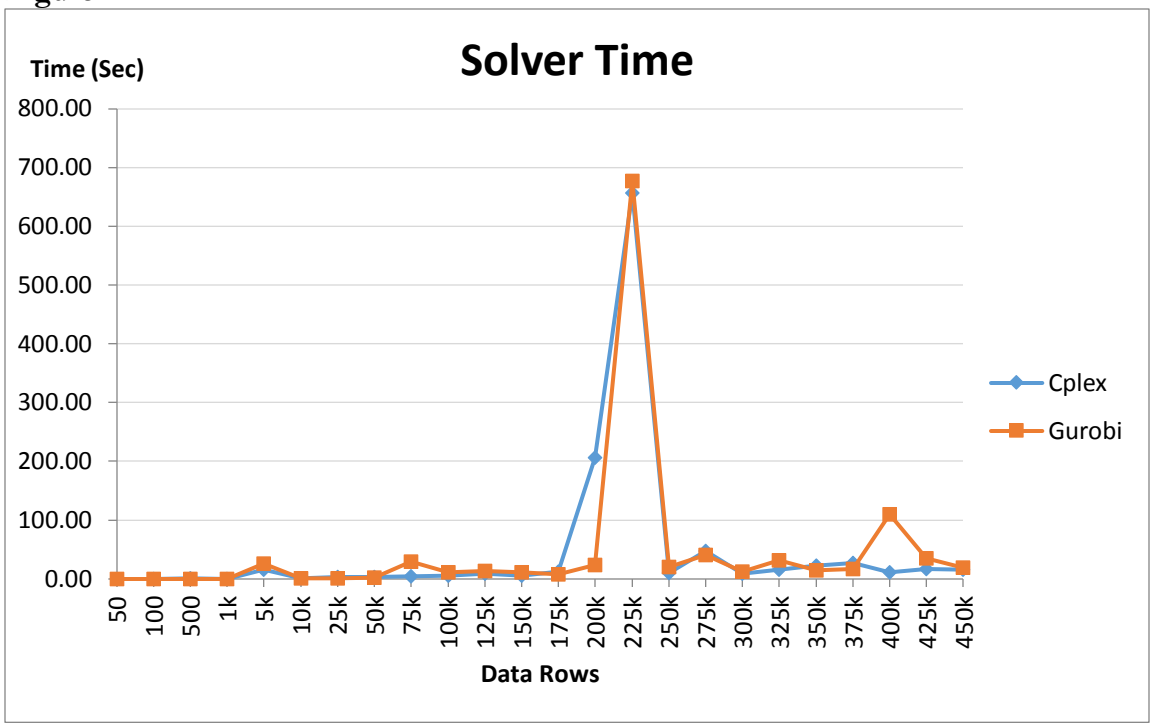


Figure 3

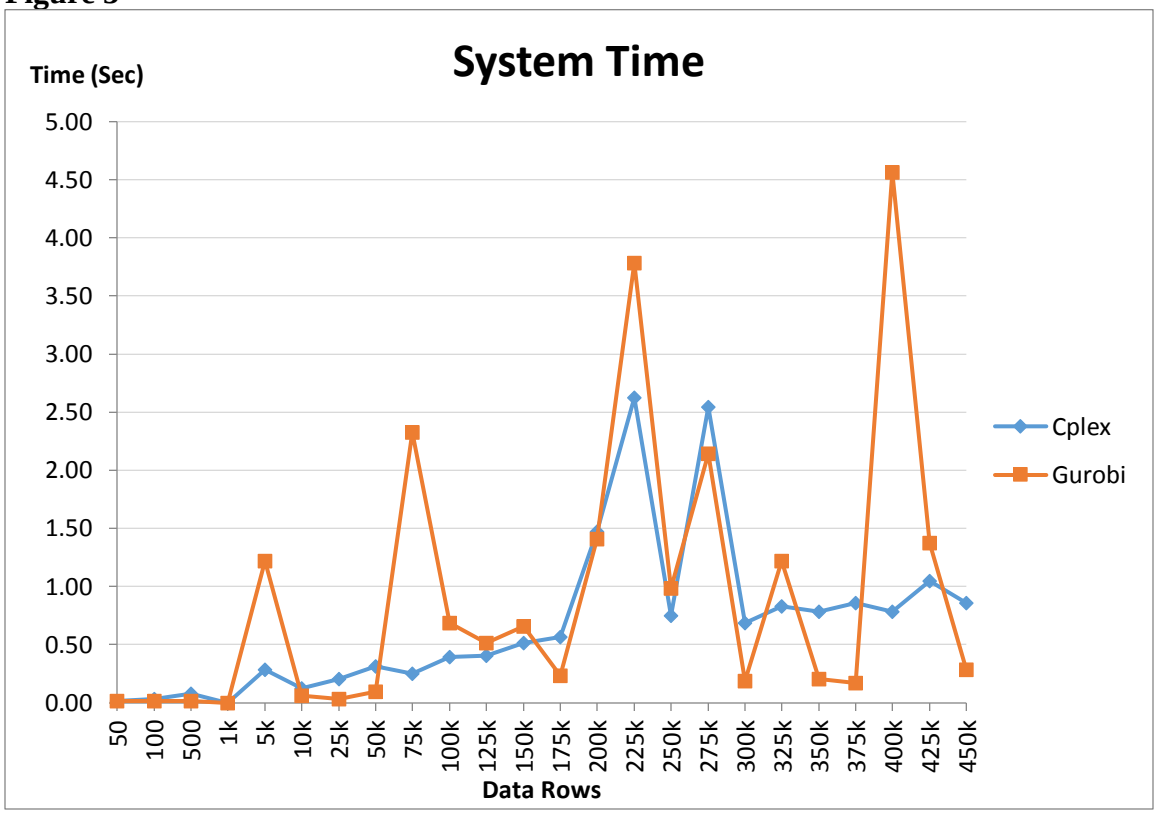


Figure 4

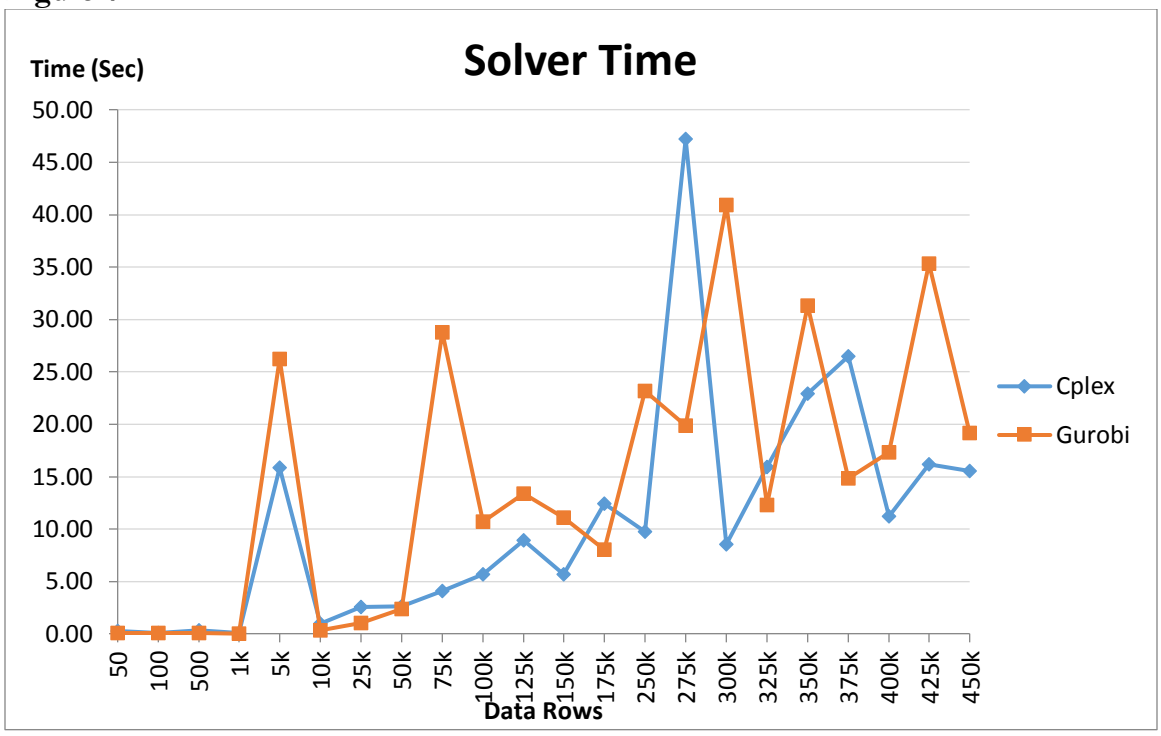


Figure 5

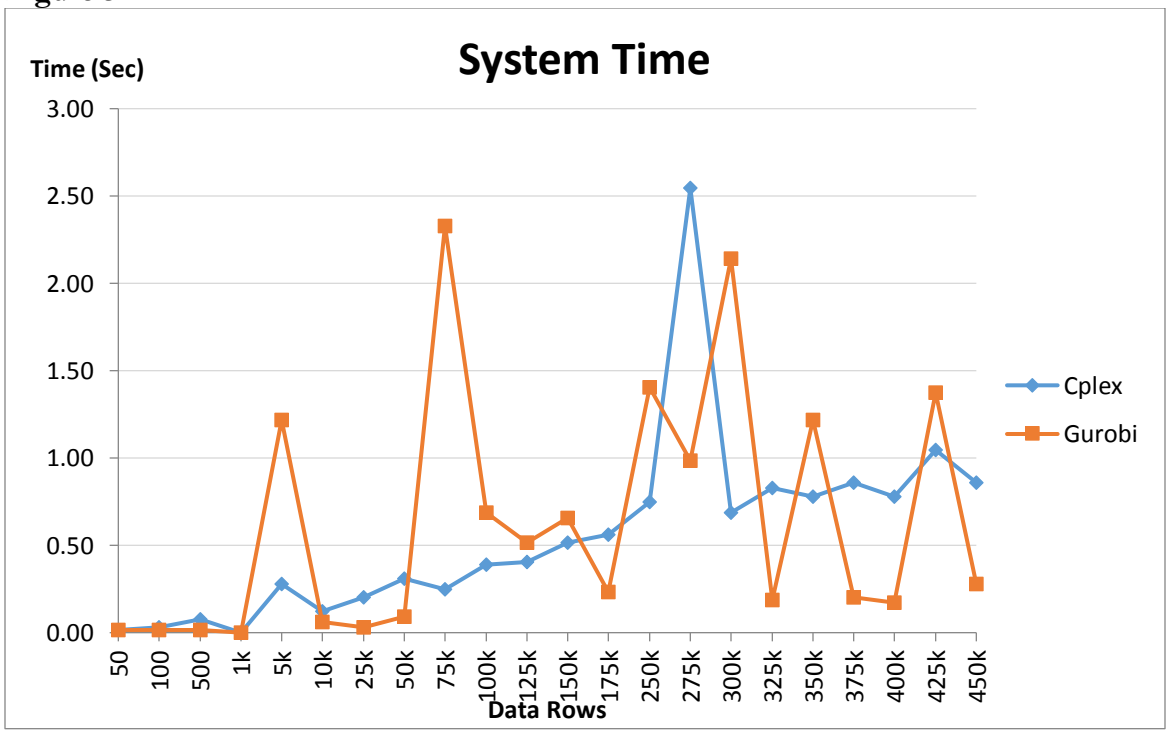
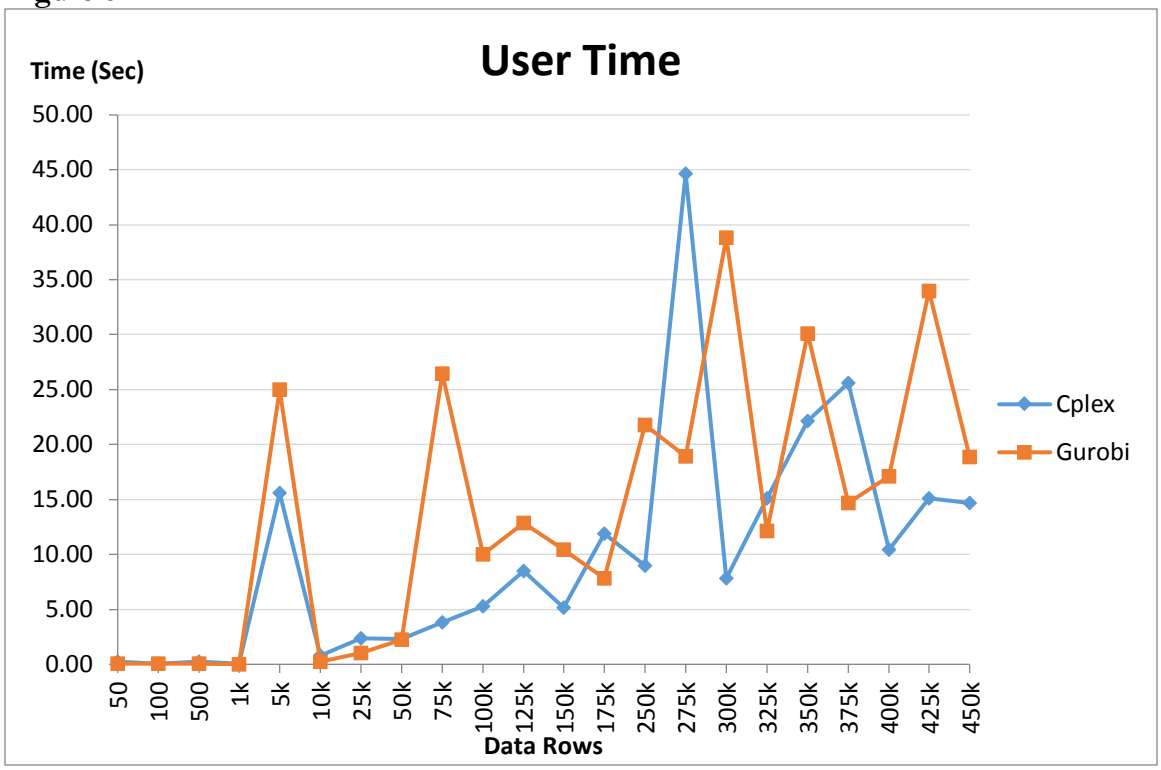


Figure 6



Finally, we tested solvers performance with a single dataset (100k) and instead of increasing rows to the dataset, we gradually increased the value of the risk constraint. We expected to see a gradual increase in calculation times since relaxing the constraint would inevitably reduce the number of variables that could be ruled out with presolving. We did not see an increase in calculation times as we expected during this test, however we did notice an interesting pattern of peaks and troughs that alternated between the two solvers. See Figures 7 – 9 below for these results.

Figure 7

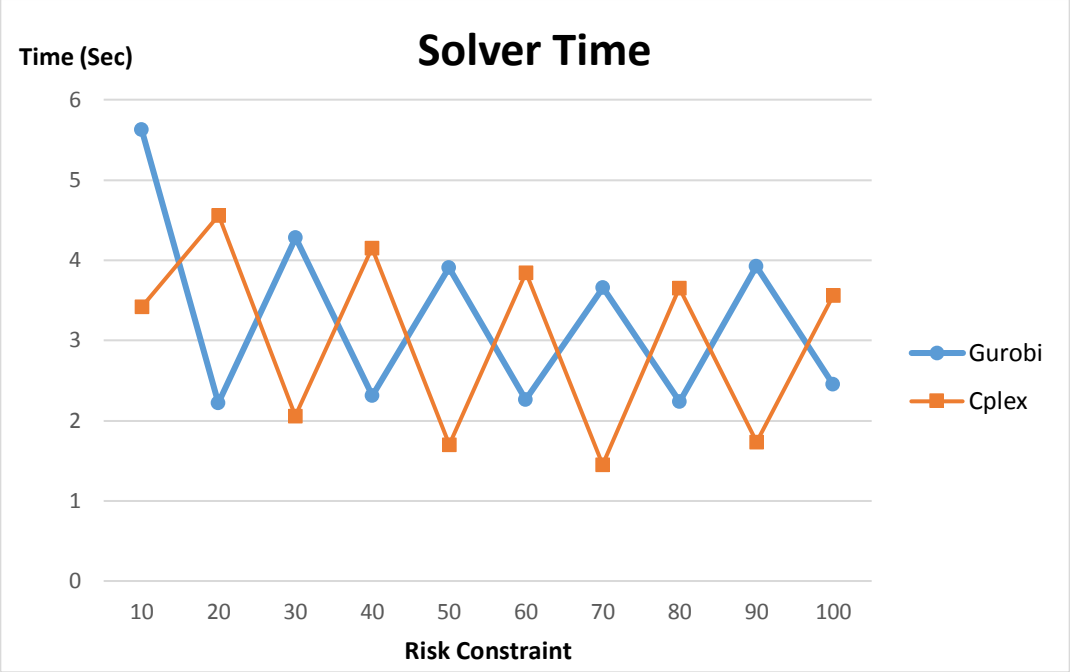


Figure 8

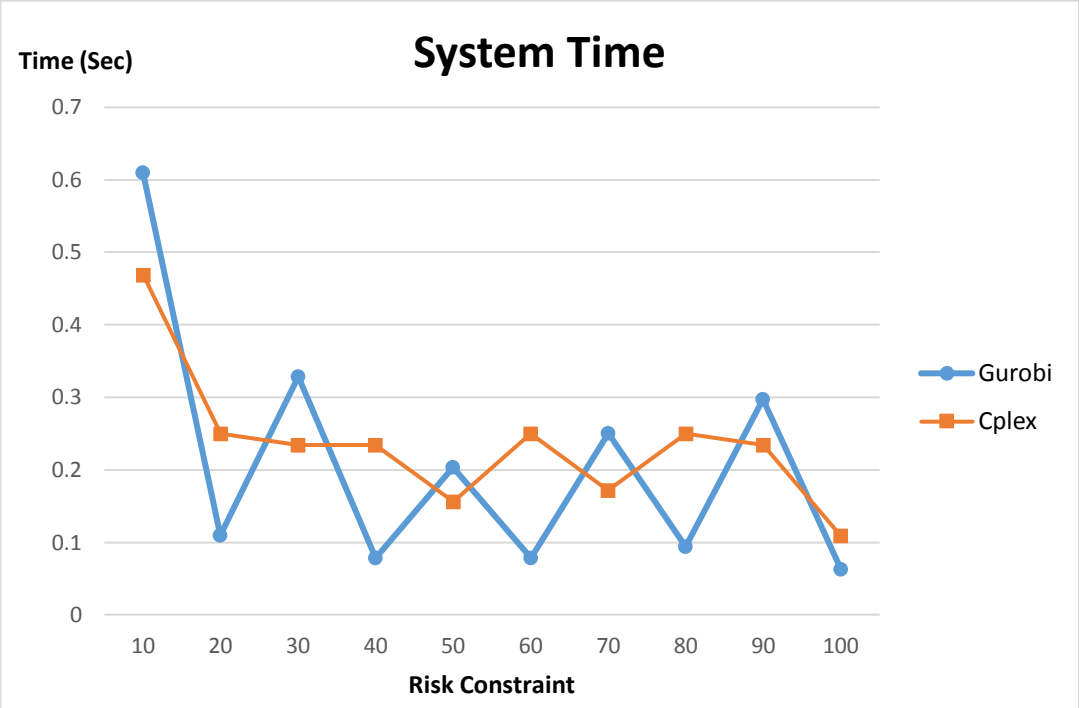
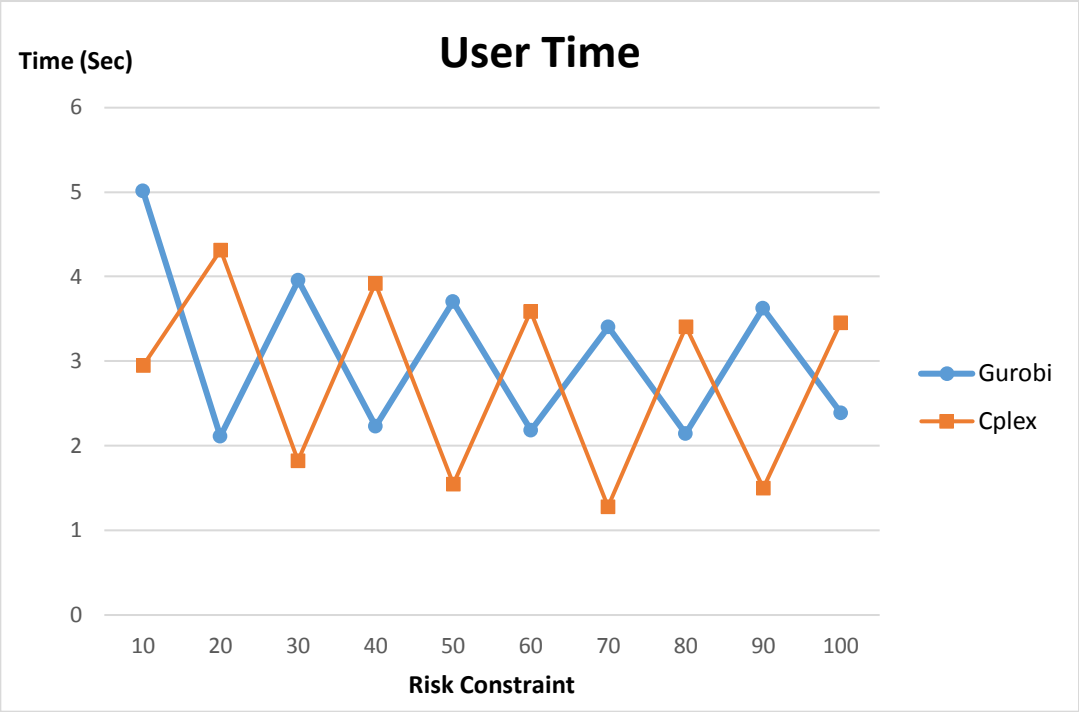


Figure 9



References

1. https://en.wikipedia.org/wiki/Knapsack_problem
2. Kellerer, Pferschy, and Pisinger 2004, p. 449
3. K. H. Kim and H.-O. Gnther, Container Terminals and Cargo Systems: Design, Operations Management and Logistics Control Issues. Springer, 2007.
4. B. Chor and R. L. Rivest, "A knapsack-type public key cryptosystem based on arithmetic infinite fields," IEEE Transactions on Information Theory, vol. 34, 1988
5. Clifford C Petersen, "Computational Experience with Variants of the Balas Algorithm applied to the selection of R&D projects," Management Science 13(9) (1967) 736-750
6. http://ampl.com/products/ampl/
7. http://lendingclub.com
8. http://www.gurobi.com/resources/getting-started/mip-basics
9. https://www.quora.com/Why-cplex-use-primal-dual-method%EF%BC%9Fdoes-IPM-faster-than-simplex-when-solving-linear-programing
10. http://www.chem.uoa.gr/applets/AppletSimplex/Appl_Simplex2.html
11. Branch and Cut algorithms for Combinatorial Optimization Problems, Avijit Sarkar
12. Branch-and-Cut Algorithms for Combinatorial Optimization Problems, John E. Mitchell, April 19, 1999, revised September 7, 1999.
13. https://en.wikipedia.org/wiki/CPLEX

Appendix 1: AMPL Model and Data File for Test Problem 1

```
set year;
set project;

param budget{i in year}> 0;
param contractvolume{j in project} > 0;
param cost {i in year, j in project} >= 0;
param solvetime = _total_solve_time >=0;
param systemtime = _total_solve_system_time >=0;
param usertime = _total_solve_user_time >=0;

var x{j in project} binary;

maximize zmax: sum {j in project} c[j]*x[j];

subject to yearlybudget {i in year}:
  sum{j in project} cost[i,j]* x[j]  <= budget[i];
```

```
set project := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28;
set year:= a1 a2 a3 a4 a5 a6 a7 a8 a9 a10;
param contractvolume :=
1 100       8 120       15 650      22 1100
2 220       9 160       16 320      23 950
3 90        10 580      17 480      24 450
4 400       11 400      18 80       25 300
5 300       12 140      19 60       26 220
6 400       13 100      20 2550     27 200
7 205       14 1300     21 3100     28 520;

param budget :=
a1 930      a6 572
a2 1210     a7 240
a3 272      a8 400
a4 462      a9 470
a5 532      a10 490;

param cost:
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28:=
a1 8 24 13 80 70 80 45 15 28 90 130 32 20 120 40 30 20 6 3 180 220 50 30 50 12 5 8
18
a2 8 44 13 100 100 90 75 25 28 120 130 32 40 160 40 60 55 10 6 240 290 80 90 70 27
17 8 28
a3 3 6 4 20 20 30 8 3 12 14 40 6 3 20 5 0 5 3 0 20 30 40 10 0 5 0 0 10
a4 5 9 6 40 30 40 16 5 18 24 60 16 11 30 25 10 13 5 1 80 60 50 20 30 10 5 3 20
a5 5 11 7 50 40 40 19 7 18 29 70 21 17 30 25 15 25 5 1 100 70 55 20 50 15 15 6 20
a6 5 11 7 55 40 40 21 9 18 29 70 21 17 35 25 20 25 5 2 110 70 55 20 50 20 15 6 20
a7 0 0 1 10 4 10 0 6 0 6 32 3 0 70 10 0 0 0 0 0 30 10 0 10 10 5 0 10
a8 3 4 5 20 14 20 6 12 10 18 42 9 12 100 20 5 6 4 1 20 50 30 5 20 20 10 10 20
a9 3 6 9 30 29 20 12 12 10 30 42 18 18 110 20 15 18 7 2 40 60 50 25 25 25 15 10 28
a10 3 8 9 35 29 20 16 15 10 30 42 20 18 120 20 20 22 7 3 50 60 55 25 30 25 15 10
28;
```

Appendix 2: AMPL Model File for Test Problem 2. The AMPL data file is the same as Test

```
set year;
set project;

param budget{i in year}> 0;
param contractvolume{j in project} > 0;
param cost {i in year, j in project} >= 0;
param solvetime = _total_solve_time >=0;
param systemtime = _total_solve_system_time >=0;
param usertime = _total_solve_user_time >=0;

var x{j in project} binary;

maximize zmax: sum {j in project} contractvolume[j]*x[j];

subject to yearlybudget {i in year}: sum{j in project} cost[i,j]* x[j]  <=
budget[i];
# Mutually Exclusive Constraints
subject to mc1:x[4] + x[5] <=1;
subject to mc2:x[8] + x[11]<=1;
subject to mc3:x[9] + x[14]<=1;
subject to mc4:x[17]+ x[18]<=1;
subject to mc5:x[22]+ x[23]<=1;
subject to mc6:x[26]+ x[28]<=1;
# Dependent Constraints
subject to dc1:x[4] <= x[3];
subject to dc2:x[5] <= x[3];
subject to dc3:x[6] <= x[3];
subject to dc4:x[7] <= x[3];
subject to dc5:x[11]<= x[2];
subject to dc6:x[24]<= x[23];
subject to dc7:x[20]<= x[19];
```

Appendix 3: AMPL Model and Data File for Test Problem 7

```
set J ; # number j^th PROJECT
set I ; # time periods

param solvetime = _total_solve_time >= 0; # used to display solver duration
param systemtime = _total_solve_system_time >= 0;# used to display solver duration
param usertime = _total_solve_user_time >= 0; # used to display solver duration
param volume {j in J} >= 0;#allow profit to change with number  of projects
param cost {i in I, j in J} >= 0; # Cost for individual projects
param budget {i in I} >= 0; # Resource Limit i^th year
var Choose {j in J} binary; # Logical Constraint

maximize contract_volume : #Objective function
sum{j in J} volume[j] * Choose[j] ;
subject to AnnualResource {i in I} : # Constraint
sum {j in J} cost[i,j] * Choose[j] <= budget[i] ;
```

```
set J:=
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,3
1,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50 ;

set I:= 1,2,3,4,5 ;

param volume:=
1 560 2 1125 3 300 4 620 5 2100 6 431 7 68 8 328 9 47 10 122 11 322 12 196 13 41
14 25 15 425 16 4260 17 416 18 115 19 82 20 22 21 631 22 132 23 420 24 86 25 42 26
103 27 215 28 81 29 91 30 26 31 49 32 420 33 316 34 72 35 71 36 49 37 108 38 116
39 90 40 738 41 1811 42 430 43 3060 44 215 45 58 46 296 47 620 48 418 49 47 50 81;

param cost (tr) :
      1    2    3    4  5 :=
 1   40   16   38    8    38
 2   91   92   39   71    52
 3   10   41   32   30    30
 4   30   16   71   60    42
 5  160  150   80  200   170
 6   20   23   26   18     9
 7    3    4    5    6     7
 8   12   18   40   30    20
 9    3    6    8    4     0
10   18    0   12    8     3
11    9   12   30   31    21
12   25    8   15    6     4
13    1    2    0    3     1
```

```
…continued param cost (tr)

   14     1     1     1     0     2
   15    10     0    23    18    14
   16   280   200   100    60   310
   17    10    20     0    21     8
   18     8     6    20     4     4
   19     1     2     3     0     6
   20     1     1     0     2     1
   21    49    70    40    32    18
   22     8     9     6    15    15
   23    21    22     8    31    38
   24     6     4     0     2    10
   25     1     1     6     2     4
   26     5     5     4     7     8
   27    10    10    22     8     6
   28     8     6     4     2     0
   29     2     4     6     8     0
   30     1     0     1     0     3
   31     0     4     5     2     0
   32    10    12    14     8    10
   33    42     8     8     6     6
   34     6     4     2     7     1
   35     4     3     8     1     3
   36     8     0     0     0     0
   37     0    10    20     0     3
   38    10     0     0    20     5
   39     1     6     0     8     4
   40    40    28     6    14     0
   41    86    93    12    20    30
   42    11     9     6     2    12
   43   120    30    80    40    16
   44     8    22    13     6    18
   45     3     0     6     1     3
   46    32    36    22    14    16
   47    28    45    14    20    22
   48    13    13     0    12    30
   49     2     2     1     0     4

   50     4     2     2     1    0;
  param budget := 1 800 2 650 3 550 4 550 5 650;
```

Appendix 4: Rollover: AMPL Model File for Test Problem 7. The AMPL mod file

```
set J :=
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,3
0,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50 ;

set I := 1,2,3,4,5 ;

param volume:=
1 560 2 1125 3 300 4 620 5 2100 6 431 7 68 8 328 9 47 10 122
11 322 12 196 13 41 14 25 15 425 16 4260 17 416 18 115 19 82 20 22 21 631 22
132 23 420 24 86 25 42 26 103 27 215 28 81 29 91 30 26
31 49 32 420 33 316 34 72 35 71 36 49 37 108 38 116 39 90 40 738
41 1811 42 430 43 3060 44 215 45 58 46 296 47 620 48 418 49 47 50  81;

param cost:=
      1    2    3    4  :=
 1   40   56   38   46   84
 2   91  183   39  110  162
 3   10   51   32   62   92
 4   30   46   71  131  173
 5  160  310   80  280  450
 6   20   43   26   44   53
 7    3    7    5   11   18
 8   12   30   40   70   90
 9    3    9    8   12   12
10   18   18   12   20   23
11    9   21   30   61   82
12   25   33   15   21   25
13    1    3    0    3    4
14    1    2    1    1    3
15   10   10   23   41   55
16  280  480  100  160  470
17   10   30    0   21   29
18    8   14   20   24   28
19    1    3    3    3    9
20    1    2    0    2    3
21   49  119   40   72   90
22    8   17    6   21   36
23   21   43    8   39   77
24    6   10    0    2   12
25    1    2    6    8   12
```

```
…continued param cost (tr)

26    5   10    4   11   19
27   10   20   22   30   36
28    8   14    4    6    6
29    2    6    6   14   14
30    1    1    1    1    4
31    0    4    5    7    7
32   10   22   14   22   32
33   42   50    8   14   20
34    6   10    2    9   10
35    4    7    8    9   12
36    8    8    0    0    0
37    0   10   20   20   23
38   10   10    0   20   25
39    1    7    0    8   12
40   40   68    6   20   20
41   86  179   12   32   62
42   11   20    6    8   20
43  120  150   80  120  136
44    8   30   13   19   37
45    3    3    6    7   10
46   32   68   22   36   52
47   28   73   14   34   56
48   13   26    0   12   42
49    2    4    1    1    5
50    4    6    2    3    3

param budget :=
     1 800 2 1450 3 550 4 1100 5 1750;
```

Appendix 5: Loans: AMPL Model File for Loans Model. The AMPL mod file

```
set Loans;
set Year;

param principal {Loans} >= 0;
param profit {Loans} >= 0;
param risk {Loans,Year} >= 0;
param budget {Year} >= 0;
param maxRisk {Year} >=0;

param solvetime = _total_solve_time >=0;
param systemtime = _total_solve_system_time >=0;
param usertime = _total_solve_user_time >=0;

var Lend {i in Loans} binary;

maximize Total_Profit:
     sum {i in Loans} Lend[i] * profit[i];

subject to Budget {t in Year}:
     sum {i in Loans} Lend[i] * principal[i] <= budget[t];

subject to Annual_Risk {t in Year}:
     sum {i in Loans} (Lend[i] * risk[i,t]) <= maxRisk[t];
```

Appendix 6: Loans: AMPL Data File for Loans Model. The AMPL dat file

```
set Year := Y1 Y2 Y3 Y4 Y5;

param: Loans:  principal profit :=
          68426699  20000     2064.04
          67275481  20000     2725.36
          68446771  7200      1883.52
          68615044  16000     6332.60
          68466995  10000     1266.20
          68516838  23850     11922.60
          68526907  16000     8362.40
          68617034  14650     8883.80
          68356614  7200      1344.96
          67849662  4225      1036.76
          68565856  9000      1329.12
          68466066  20200     10901.00
          68385794  15000     2622.72
          68446769  7000      837.92
          68566925  11000     1624.12
          68476807  10400     6994.60
          68341789  24250     17810.60
          68606528  12000     1771.80
          68587709  21000     7953.60
          68506885  10000     3344.00
          68487261  4200      926.76
          68416935  15000     2373.96
          68516507  14000     2215.48
          68376217  23100     14007.60
          68526942  27300     7872.00
          68446093  11550     5520.60
          68377006  6000      759.72
          68436917  16000     5784.20
          68597047  19000     8031.20
          68356421  22400     8098.00
          68416256  5000      791.32
          68407301  27500     11623.60
          68356922  4200      967.08
          68416953  1500      154.92
          68587652  25000     2103.32
          68547583  8650      728.00
          68566886  29900     10809.40
          68537594  5000      633.28
          68366999  15850     11030.60
          66796130  8800      1392.68
          68506798  23000     5306.20
          68466926  10000     1032.20
```

```
…continued param: Loans: principal profit :=
          68354783  9600 1148.88
          68607141  17600     1481.08
          68426545  16000     5784.20
          68527009  20000     3164.92
          68397043  28000     11835.20
          68585839  17475     4023.12
          68466922  17925     8960.40;

param risk:    Y1   Y2   Y3   Y4   Y5 :=
     68426699  1642.00   1094.67   547.33    0.00 0.00
     67275481  2734.00   1822.67   911.33    0.00 0.00
     68446771  1790.64   1193.76   596.88    0.00 0.00
     68615044  3342.40   2673.92   2005.44   1336.96   668.48
     68466995  963.00    642.00    321.00    0.00 0.00
     68516838  6289.25   5031.40   3773.55   2515.70   1257.85
     68526907  4331.20   3464.96   2598.72   1732.48   866.24
     68617034  4359.84   3487.87   2615.90   1743.94   871.97
     68356614  1199.52   799.68    399.84    0.00 0.00
     67849662  918.94    612.63    306.31    0.00 0.00
     68565856  1291.50   861.00    430.50    0.00 0.00
     68466066  5573.18   4458.54   3343.91   2229.27   1114.64
     68385794  2394.00   1596.00   798.00    0.00 0.00
     68446769  644.70    429.80    214.90    0.00 0.00
     68566925  1578.50   1052.33   526.17    0.00 0.00
     68476807  3593.20   2874.56   2155.92   1437.28   718.64
     68341789  8812.45   7049.96   5287.47   3524.98   1762.49
     68606528  1722.00   1148.00   574.00    0.00 0.00
     68587709  4271.40   3417.12   2562.84   1708.56   854.28
     68506885  1889.00   1511.20   1133.40   755.60    377.80
     68487261  854.28    569.52    284.76    0.00 0.00
     68416935  2247.00   1498.00   749.00    0.00 0.00
     68516507  2097.20   1398.13   699.07    0.00 0.00
     68376217  6874.56   5499.65   4124.74   2749.82   1374.91
     68526942  7199.01   4799.34   2399.67   0.00 0.00
     68446093  2967.20   2373.76   1780.32   1186.88   593.44
     68377006  577.80    385.20    192.60    0.00 0.00
     68436917  3164.80   2531.84   1898.88   1265.92   632.96
     68597047  4132.50   3306.00   2479.50   1653.00   826.50
     68356421  4430.72   3544.58   2658.43   1772.29   886.14
     68416256  749.00    499.33    249.67    0.00 0.00
     68407301  5981.25   4785.00   3588.75   2392.50   1196.25
```

```
…continued param risk:   Y1   Y2   Y3   Y4   Y5 :=
     68356922  877.38    584.92    292.46    0.00 0.00
     68416953  123.15    82.10     41.05     0.00 0.00
     68587652  1760.00   1173.33   586.67    0.00 0.00
     68547583  608.96    405.97    202.99    0.00 0.00
     68566886  5914.22   4731.38   3548.53   2365.69   1182.84
     68537594  481.50    321.00    160.50    0.00 0.00
     68366999  5583.96   4467.16   3350.37   2233.58   1116.79
     66796130  1318.24   878.83    439.41    0.00 0.00
     68506798  3144.10   2515.28   1886.46   1257.64   628.82
     68466926  821.00    547.33    273.67    0.00 0.00
     68354783  884.16    589.44    294.72    0.00 0.00
     68607141  1239.04   826.03    413.01    0.00 0.00
     68426545  3164.80   2531.84   1898.88   1265.92   632.96
     68527009  2996.00   1997.33   998.67    0.00 0.00
     68397043  6090.00   4872.00   3654.00   2436.00   1218.00
     68585839  3650.53   2433.69   1216.84   0.00 0.00
     68466922  4726.82   3781.46   2836.09   1890.73   945.36;

param: budget :=
          Y1   100000
          Y2   125000
          Y3   90000
          Y4   110000
          Y5   130000;

param: maxRisk :=
          Y1   30000
          Y2   25000
          Y3   20000
          Y4   18000
          Y5   16000;
```