

## **A NOVEL COMPARISON OF KNAPSACK PROBLEM COMPUTATION TIME IN PYTHON AND R**

PROJECT FOR OPER 610: LINEAR PROGRAMMING

Matthew D. Ferguson, Major, U.S. Army  
AFIT-ENS-17-M-1

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

**AIR FORCE INSTITUTE OF TECHNOLOGY**

---

Wright-Patterson Air Force Base, Ohio

## Project Objectives:

The objectives of this project are to practice to utilizing different computer programs to solve mathematical programming problems, demonstrate familiarity with the testing and comparison of solution methods of those programs, become familiar with technical writing and exhibit intellectual curiosity.

## Model, and Method to Generate Instances:

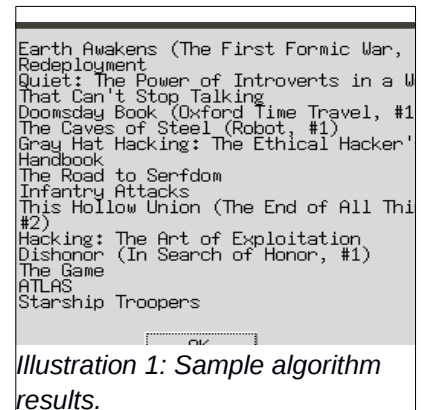
This project studies computational time in addressing the vexful question, "what should I read next?". Given a comma separated value (csv) file from a Goodreads library and an objective derived from the desire to make the most out of limited reading time, we can formulate a mixed integer linear program with boolean decision variables to optimize reading time subject to some desired constraints. The resultant output should provide insight into what books to shelve or acquire for consumption over the next time period, in this case a fiscal quarter. A sample of such an output is shown in Illustration 1.

Given the structure of this model, we can utilize the formulation of the knapsack problem. A knapsack is a combinatorial optimization problem representing the decision to choose a limited subset of a collection of objects, each with some known utility and cost, such taking up space or weight in the knapsack. Wikipedia cites the boolean knapsack as the most common formulation of the knapsack problem, where the number of copies of each kind of item to zero or one. Given a set of  $n$  items numbered from 1 up to  $n$ , each with a weight  $w_i$  and a value  $v_i$ , along with a maximum weight capacity  $W$ :

$$\text{Maximize } \sum_{i=1}^n v_i x_i \text{ s.t. } \sum_{i=1}^n w_i x_i \leq W \text{ and } x_i \in \{0, 1\} \quad (1)$$

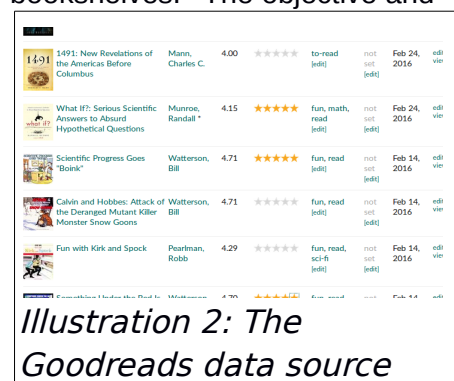
For this problem, the decision variables consist of the [then] 431 books from an online database record, specifically from the Goodreads library of the author. These titles are further divided among several not mutually exclusive categories or 'bookshelves.' The objective and technological coefficients come from ratings and other data in this database. As shown in Illustration 2, the database provides ratings and other information about each book. The value of each boolean decision variable can be interpreted logically as either a true or false statement regarding whether that book should be read during the time period under consideration, and there are multiple weight dimensions ( $W$ ) to satisfy.

The objective is to maximize the product of page count and either the imputed ratings of the user if available, or the



Earth Awakens (The First Formic War, Redeployment)  
Quiet: The Power of Introverts in a World That Can't Stop Talking  
Doomsday Book (Oxford Time Travel, #1)  
The Caves of Steel (Robot, #1)  
Gray Hat Hacking: The Ethical Hacker's Handbook  
The Road to Serfdom  
Infantry Attacks  
This Hollow Union (The End of All Things, #2)  
Hacking: The Art of Exploitation  
Dishonor (In Search of Honor, #1)  
The Game  
ATLAS  
Starship Troopers

Illustration 1: Sample algorithm results.



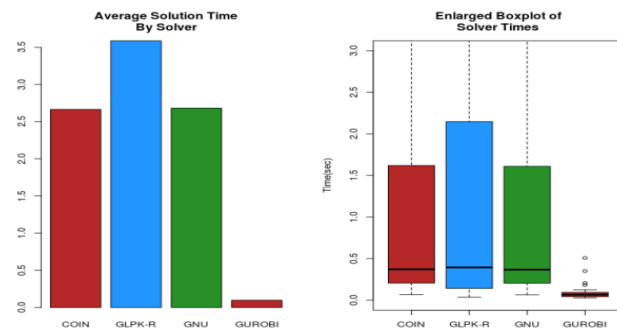
Book Cover	Title	Author	Rating	Category	Date	Options
	1491: New Revelations of the Americas Before Columbus	Mann, Charles C.	4.00	to-read	Feb 24, 2016	edit view
	What If?: Serious Scientific Answers to Absurd Hypothetical Questions	Murrow, Randall	4.15	fun, math, read	Feb 24, 2016	edit view
	Scientific Progress Goes to Town	Watterson, Bill	4.71	fun, read	Feb 14, 2016	edit view
	Calvin and Hobbes: Attack of the Deranged Mutant Killer Monster Snow Goons	Watterson, Bill	4.71	fun, read	Feb 14, 2016	edit view
	Fun with Kirk and Spock	Pearlman, Robb	4.29	fun, read, sci-fi	Feb 14, 2016	edit view

Illustration 2: The Goodreads data source

average Goodreads community rating if there is no user assigned rating. This product of scalar values yields a linear objective function and adequately weights the preferences of the decision-maker. Technological coefficients stem from the weighted preferences of the decision-maker regarding various constraints, such as either the percentage or absolute number of books from a particular shelf. In order to develop a test set of 30 model instances, we utilized the numpy Python package to generate random variates for each parameter, either as integers for the available number of hours or specific book category limits, or real valued percentages for percentage constraints related to proportion of reading from certain bookshelves.

In order to construct the model, we made use of the PuLP python package, which utilizes the APIs of various solvers to provide a stable and portable call across solver platforms. The PuLP package also provides a high-level programming interface to construct the model in an intuitive format. While PuLP can generate MPS and LP files for models once they are constructed, PuLP cannot read those files back into the PuLP framework. This limitation had impacts during the second experiment of our study.

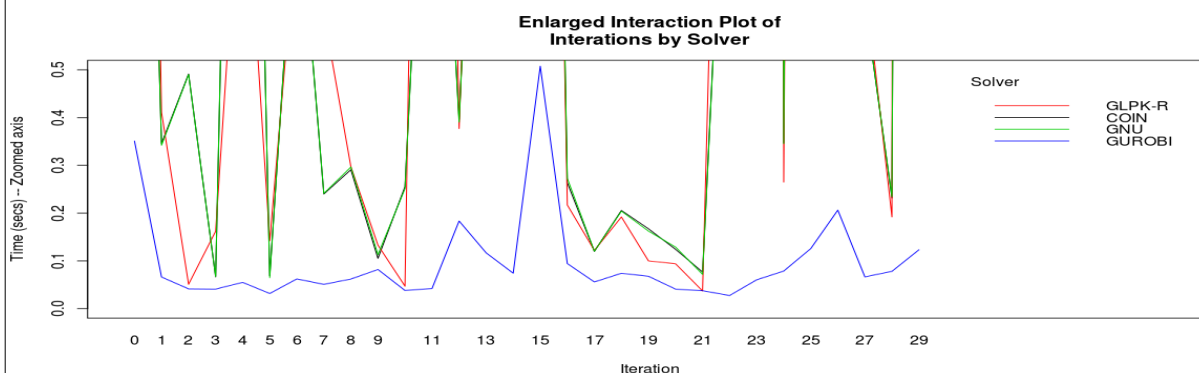
The solvers were installed or compiled on a 64 bit Debian Linux kernel (Ubuntu 12.04LTS) running on an Intel i5 quadcore 2.53 GHz CPU with 4 GB of memory. For the first experiment, the Python environment was an Anaconda Python 2.7 Installation. For the second experiment we also utilized a commercially modified Gurobi Python 2.7 environment, which is packaged with the Gurobi solver.



*Illustration 3: Average and Median Solution Times by Solver*

### First Experiment:

Our first experiment involved examining the differences across a variety of available Solvers. For this study, we included the CBC-COIN(COIN) solver included with the PuLP python package as well as the GNU Linear Programming Kit (GLPK, herein noted GNU) solver and the commercial solver Gurobi. In order to explore some other programming languages, we also



*Illustration 4: Gurobi demonstrates first order stochastic dominance*

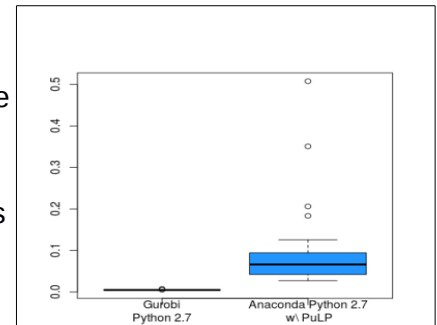
examined the performance of an R-Programming package that also utilizes the GLPK solver (GLPK-R). We attempted to include both CPLEX and an R programming package that could interface with the lpSolve library, however certain technical limitations force any consideration of these options to occur in subsequent analysis. We utilized the timeit python function to measure computational effort as processor time.

As shown in Illustration 3, the Gurobi solver performed the fastest, with regard to average time. Illustration 4 demonstrates that while there are some Gurobi iterations that are slower than other solvers, when taking into account the interaction of solver and Iteration, Gurobi is at least as fast as any other solver and stochastically dominates the other options.

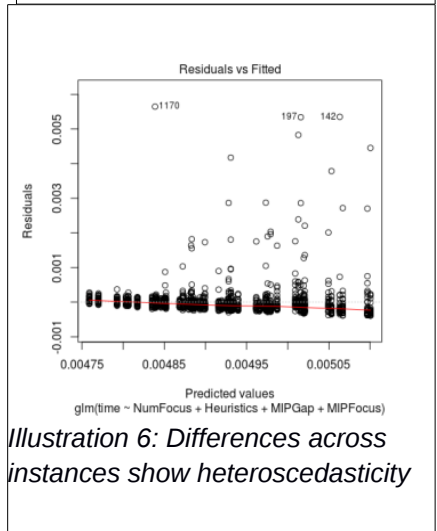
We further examined the simultaneous difference in means between solvers by modeling solver and iteration as factors. We then examined simultaneous contrasts using both Scheffe and Tukey. Scheffe is useful in examining unplanned contrasts and in general is extremely conservatives, but given equal observations within groups, Tukey provides the best results for pairwise comparisons. Given both methods agreement, we can be confident that there exists a true difference in computation time between the GLPK solver accessed from R and Gurobi. Given the large differences in solution time variances between iteration, we require more data before we can state with great confidence that there exists a true difference between Gurobi and the remaining solvers, however that data appears highly suggestive that this is true.

Second Experiment: Our second experiment examined the impact of changes to several of Gurobi's default parameters on the solution time of the same test set of thirty boolean knapsack problems. We altered the MIP focus, the solution gap tolerance, the percentage of time spent using heuristics, and a Numerical Focus parameter. <sup>ii</sup> The first observation, as shown in Illustration 5 is that run-times were reduced with statistical significance in Gurobi's python environment than when compared to the standard Anaconda environment using PuLP. However, as a convenience we altered Gurobi's parameters in the Gurobi shell since PuLP lacks the ability to reconstitute PuLP models from either .MPS or .LP files. Results were again compared using both Scheffe and Tukey, with agreement in the results.

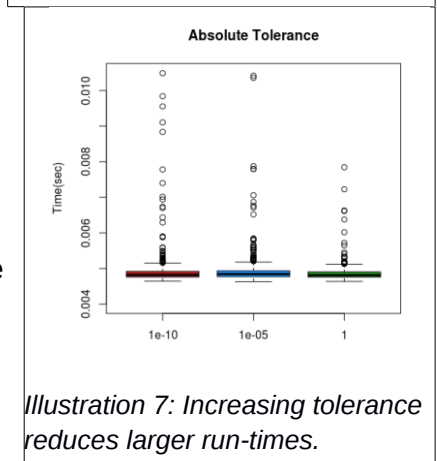
Illustration 6 highlights the heteroscedasticity present across observations, confirming again that some iterations are simply more computationally demanding than others. Illustration 7



*Illustration 5: Not all python environments are equal for Gurobi*



*Illustration 6: Differences across instances show heteroscedasticity*



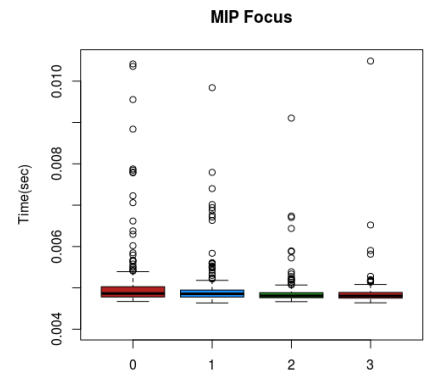
*Illustration 7: Increasing tolerance reduces larger run-times.*

shows the somewhat intuitive result that increasing the gap tolerance allows more computationally difficult iterations to be solved faster, at the cost of solution quality. MIP focus changes, as shown in Illustration 8, show the counter-intuitive result that any focus chosen performs faster than the default setting of automatic.

**Conclusions:** This project appears to confirm some of the findings of Mittelman<sup>iii</sup> regarding MILP performance between, among other solvers, Gurobi and the CBC(COIN) solver. Furthermore, we've shown that there is a statistical difference between using the Gurobi Python environment and simply calling Gurobi from another. Further work might show whether this additional time is some constant associated with the API call or whether the delay might scale with increasingly large problems.

We've also shown that for the problem examined, we can tune Gurobi's parameters for optimal speed, especially if we find small magnitude differences (i.e. a single star-page) acceptable in a high-quality solution and the absolute optimal.

This work also raises some other questions for future investigation. Is the slow performance of the automatic (0) MIP Focus setting purely coincidental or related to problem size or structure? Other structures and sizes could be examined to explore this phenomena, and potentially result in a recommendation to not use the automatic MIP Focus setting in certain cases. Also, we could extend the first part of our work into other modeling approaches and extending the list of solvers to include CPLEX and the LPSolve library. Future work could compare python's Pyomo, PyLPSolve, and CVXOPT, R's LpSolveAPI and Adagio (with a specialized knapsack function) packages to more traditional modeling approaches such as Julia, GAMS and AMPL. Do certain approaches and wrapper libraries impose a singular translation cost to call a particular solver, or do some modeling frameworks impose a level of bloat that might make them unusable for problems over a certain size? Increased data on the original solvers might be able to overcome the different variances of each iteration to show other mean differences in processor time as statistically significant. These are all open questions that are critical to understanding the choice of tool for a large-scale optimization problem.



*Illustration 8: MIP Focus Settings, anything is better than automatic.*

i Knapsack problem. n.d. In *Wikipedia*. Retrieved February 14, 2016. From [https://en.wikipedia.org/wiki/Knapsack\\_problem#Definition](https://en.wikipedia.org/wiki/Knapsack_problem#Definition)

ii Parameters. n.d. In *Gurobi Optimization Reference Manual*. Retrieved February 19, 2016 from <https://www.gurobi.com/documentation/6.5/refman/parameters.html#sec:Parameters>

iii Mittelman, H. 6 Dec 2015. "Mixed Integer Linear Programming Benchmark (MILP2010)". Retrieved March 7, 2016 from <http://plato.asu.edu/ftp/milpc.html>