CS164 Final Project

Dennis Kageni

April 23, 2020

1 Problem Description:

Pigeon Shopping is an application (like Glovo and Instacart) that I've been working with which provides grocery delivery services to residents in Nairobi. The bulk of the business value is driven by increasing the margins and volume of grocery delivery orders made via the platform.

Deliveries are done by motorcycle delivery partners who carry a max weight of 9 kg worth of shopping orders in carrier bags. Owing to the COVID-19 crisis, the Pigeon team has been collecting data on the most frequently bought products in the month of April. We want use this information to create a 'COVID-19 Essentials catalogue on the application's home screen.

Task: Pigeon wants to curate a catalogue of products from our record of the most bought products in April that maximizes the our revenue but does not exceed a weight of 9 kilograms (the maximum weight we can carry on a motorcycle per order) or volume of 48000cm^3 (the volume of our rider partners delivery bags').

1.1 Problem Formulation:

The problem at hand is a 0-1 Knapsack problem. This is a combinatorial optimization problem that consists of finding a solution through a discrete or finite set of solutions. The problem can be solved in a couple of ways; The greedy algorithm, via dynamic programming or as a non-convex optimization problem.

1.1.1 Integer Programs:

(Linear) Integer Programming problems are of the form:

$$\begin{aligned}
&\text{Maximize } \textstyle\sum_{j=1}^{n} c_j x_j \\
&\text{subject to:} \\
&\quad \sum_{j=1}^{n} a_{ij} x_j = b_i (i = 1, 2, \ldots, m) \\
&\qquad x_j \geq 0 (j = 1, 2, \ldots, n) \\
&\quad x_j \text{ integer } \text{ (for some or all } j = 1, 2, \ldots, n)
\end{aligned}$$

They are said to be a *mixed* integer program when some, but not all, variables are restricted to be integer, and is called a pure integer program when all decision variables must be integers (Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1992).)

In many applications, integrality restrictions reflect natural indivisibility of the problems/variables under study. This emphasises the importance of integer variables in providing broad modeling capabilities beyond those available in linear programming.

1.1.2 The Knapsack Problem:

The 0–1 Knapsack Problem (KP) is one of the paradigmatic problems in combinatorial optimization where a set of items with given profits and weights is available and the aim is to select a subset of the items in order to maximize the total profit without exceeding a known knapsack capacity.(Della Croce, F., Salassa, F., & Scatamacchia, R. (2017)). The problem is of the form:

$$\begin{aligned} &\text{Maximize} && \textstyle\sum_{k=1}^{N} r_k x_k \\ &\text{Subject to:} && \\ & && \textstyle\sum_{k=1}^{N} w_k x_k \leq c \end{aligned}$$

where;

$x_1, x_2, \ldots, x_N$ are non-negative integer-valued decision variables.
$x_k =$ the number of type- k items that are loaded into the knapsack.

And our parameters are;

$w_k =$ the weight of each type- k item, for $k = 1, 2, \ldots, N$
$r_k =$ the value associated with each type- k item, for $k = 1, 2, \ldots, N$
$c =$ the weight capacity of the knapsack.

It is however important to note that the Knapsack Problem (A mixed Integer Program) is a non-convex program. This is because of the constraint of {0,1}. The set of the feasible region is not convex as well due to the rendered constraint - it is only feasible on some discrete points of its search space.

However, to find the best combination of binary variables (the optimal solution), the problem is relaxed, i.e {0,1}→[0,1]. In other words, the problem is "made" continuous. If then the objective function and constraints are convex, this relaxed version of the problem is also convex and can be solved in polynomial time. which allows for enumeration techniques such as the branch-and-bound algorithms to be computationally viable.

2 Solution Specification:

Integer programs can be solved in a couple of ways including Enumeration Techniques such as the Branch and Bound Method, Cutting plane techniques and/or group-theoretic techniques. The cutting-plane algorithm solves integer programs by modifying linear-programming solutions until the integer solution is obtained. In practice, the branch-and-bound procedures almost always outperforms the cutting-plane algorithm.

2.1 Branch and Bound Algorithm:

Branch-and-bound is an enumeration scheme in which we search for the optimal solution by progressively dividing the feasible region into smaller sub-regions. In doing so, a search tree, also

known as a branch-and-bound tree, is constructed where each node in the tree corresponds to a particular sub-region of the feasible region. The idea is to avoid complete enumeration by calculating a lower bound on the value of the best possible solution in the sub-region at each node. If this lower bound exceeds the objective value of a solution we have already found, then we do not need to explore this sub-region any further.(Luedtke, J. (2007) As the name suggests, branch-and-bound consists of two main actions:

Bound: Given a solution set, get an upper/lower bound estimate of the best solution that can be found in the solution set. For example, one can find an upper bound for a 0–1 knapsack problem by solving its corresponding fractional knapsack problem. Since fractional knapsack problem allows selecting a fraction of an item while 0–1 knapsack problem does not, fractional knapsack problem will always yield an equal or better objective value, which can be seen as an upper bound on the objective of the 0–1 knapsack problem.

Branch: While computing bounds on the solution set, we encounter solutions which satisfy all constraints of the problem, but not feasible because the solution values are not integers. In this case we can branch on one of the fractional value variables — splitting the current solution space into two: (in the binary variable case) one that enforces the fractional value variables to be 0, and the other enforces the fractional value variable to be 1. For example, solving the fractional knapsack problem may yield a solution that takes 50% of item 2. One can then branch on item 2's variable by splitting the solution space to either include item 2 or not include item 2.

2.2 Solution to the Pigeon Shopping Problem:

Consider the following vectors:

$$
\begin{aligned}
R &= [r_1, r_2, \ldots, r_n]^\top \\
W &= [\omega_1, \omega_2, \ldots, \omega_n]^\top \\
V &= [v_1, v_2, \ldots, v_n]^\top
\end{aligned}
$$

Where **R** is the vector of Prices(KES) of each of the products, **W** is the weight in kg's of the corresponding products and **V** is the vector of the volume of the products. *Cw* and *Cv* are the maximum weight and volume(Size of products) that can be carried by a rider respectively. On restating the problem therefore:

$$
\begin{aligned}
\text{Maximize} \quad & \textstyle\sum_{k=1}^{N} r_k x_k \\
\text{Subject to:} \quad & \\
& \textstyle\sum_{k=1}^{N} w_k x_k \leq c_w \\
& \textstyle\sum_{k=1}^{N} v_k x_k \leq c_v \\
& x_i \in \{0, 1\}
\end{aligned}
$$

2.3 Note on CVXPY Solvers:

CVXPY supports the solvers ECOS, SCS, CVXOPT, GLPK, CBC, ELEMENTAL, GUROBI, MOSEK, and Xpress.(MOS(2017))

However, CVXPY supports fewer open source *mixed-integer solvers* GLPK_MI and CBC. If you need to solve a large mixed-integer problem quickly, or if you have a nonlinear mixed-integer model, then you will need to use a commercial solver such as CPLEX, GUROBI, MOSEK, or NAG.

For our use case, we will use the GLPK_MI solver. GLPK_MI is efficient for Integer Programs that are not heavy or have plenty of variables ($\leq$ 1000). Commercial solvers like Gurobi is considered to be an industry standard and handles large variable integer programs extremely well.

3 Analysis:

(See appendix for CVXPY implementation)

3.1 Experiment 1 - Weight and Volume Constraint:

The first experiment gave a selection of 31 products for the 'COVID-19 Essentials' catalogue and realized an optimal basket value of KES 9,820. The business normally makes 10% from every basket value and has a 70-30 split on that with their delivery partners. This means the business makes KES 294.6 from one particular order and the delivery partner makes KES 687.4.

3.2 Experiment 2 - Increasing the Weight Constraint:

This experiment gave a selection of 46 products for the 'COVID-19 Essentials' catalogue and realized an optimal basket value of KES 13,406. However, the 10% is split between two delivery riders and as such the business makes 402.18 from the split order and the delivery partners make KES 469.21 each for the same distance.

3.3 Experiment 3 - No Volume Constraint:

This experiment gave a selection of 31 Products for the 'COVID-19 Essentials' catalogue and realized an optimal basket value of KES 9,820. The business made 294.6 which would exactly the same as experiment 1.

3.4 Business Recommendations

Experiment 1 gives the optimal selection (31 Products) for the Covid-19 Essentials catalogue.

For Experiment 2, although Pigeon earns more on the split order that increases the weight capacity (+KES 107.58), our delivery partners earn less for the same distance travelled. Being in a highly competitive grocery delivery space with competitors such as Glovo and other local players, retention of our delivery partners is of high priority and competitive fees for riders is a key strategy to it.

In experiment 3, For the same weight constraint increasing the volume of the bags (Removal of the volume constraint altogether) doesn't improve the results of experiment one. Therefore, should the business have to decide between getting our delivery partners larger bags or diversifying our delivery fleet such that it can carry more weight(e.g cars), the latter would be an interesting option to pursue.

4 Conclusion:

The business implication of the results are to suggest the products selected by the algorithm in the 'COVID-19' product recommendations. However, the algorithm could be explored to accommodate for more constraints that would better model the complexity of the problem. For instance,

The weight constraint may not translate one-to-one with reality as size of the products may be a reasonable consideration to accommodate when running the optimization. The business could actually consider the business implications of expanding their delivery network at a time under consideration so that a delivery order could be serviced by more then one rider (A multi-knapsack problem).

5 References:

Boulton, F. (2018, March 13). Integer Programming in Python. Retrieved from https://towardsdatascience.com/integer-programming-in-python-1cbdfa240df2

Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1992). Applied mathematical programming. Reading, Mass.: Addison-Wesley. Chapter 9, sections 9.2-9.3 http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

Calafiore, G., & Ghaoui, L. E. (2014). Optimzation models. Cambridge: Cambridge U.P. Section 15.2.3. Section 13

Della Croce, F., Salassa, F., & Scatamacchia, R. (2017). An exact approach for the 0–1 knapsack problem with setups. Computers & Operations Research, 80, 61-67.

Diamond, S and Chu, E and Boyd, S. (2014). CVXPY 0.4.11 documentation [online] http://www.cvxpy.org/en/latest/index.html

Dr Ahmad Bazzi-Convex Application: https://www.youtube.com/watch?v=5jIfgETYKRE&list=PL-DDW8QIRjNOfSM_LTe1o7-lg1PIuw-0R&index=2

Luedtke, J. (2007). Integer programming approaches for some non-convex and stochastic optimization problems (Doctoral dissertation, Georgia Institute of Technology).

Mathematical Optimization Society. (2017, September). Optima 103. Retrieved from http://www.mathopt.org/Optima-Issues/optima103.pdf

Orlin, J. B., Punnen, A. P., & Schulz, A. S. (2004). Approximate local search in combinatorial optimization. SIAM Journal on Computing, 33(5), 1201-1214

Professor Stephen Boyd-Lecture 18 | Convex OPtimization II (Stanford) https://www.youtube.com/watch?v=ORo5IU9a55s&list=PL3940DD956CDF0622&index=37

Taylor, B.W. (2019).Introduction to Management Science, Global Edition (13e) Companion Module C: Integer Programming: The Branch and Bound Method. New York: Pearson. Pages C2-C9 Retrieved from https://media.pearsoncmg.com/ph/bp/bridgepages/bp_taylor_bridgepage/taylor_13e/online_modules/Tayl

6 Appendix

6.1 Raw Data

6.2 HC and LO Application

#modeling, #rightproblem: framed a real business problem were facing at Pigeon as an optimization problem where we used the information on the most bought products in April to populate a catalogue that we can add on to the application's homepage; all while trying to get a basket of products that would generate the most revenue but still respecting the weight constraint of our delivery partners; discussed the implications of the findings of the three experiments on possible business decisions; discussed the limitations of the current implementation of the problem and potential extensions in the conclusion

#optimization, #breakitdown: formulated, explained and solved the optimization problem; investigated the nature of the computed optimal solutions with various problem parameters and discussed the solutions in terms of the potential business implications for Pigeon

#cs164-optimizationstrategy: defined a well-posed and justified implementation optimization problem and solved it with the relevant optimization strategy from the perspective of a business trying to maximize our revenue but being cognizant of the constraints around us

#cs164-nonconvexprogramming: implemented a Mixed Integer Program in CVXPY that tells Pigeon what goods to include in the COVID-19 catalogue and the associated optimum revenue from that combination of good given the weight and volume constraints defined

6.3 CVXPY implementation

[ ]:
```
## importing required libraries

import cvxpy as cp
import numpy as np
import pandas as pd
```

[ ]:
```
print (cp.installed_solvers())
```

```
['CVXOPT', 'ECOS', 'ECOS_BB', 'GLPK', 'GLPK_MI', 'OSQP', 'SCS']
```

[ ]:
```
pigeon_data = pd.read_csv("https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vTTY0UQgs9dCZrflth7q_w8Jjt3wtOWN8_9f8JqSFwbKzjmT_ca9ZdHjzkyn2Jd60UAGcstLodg3Pv1/
↪pub?output=csv", skiprows=1)
```

[ ]:
```
# peek at data
pigeon_data.head(5)
```

[ ]:

	Brand	Unnamed: 1	...	Volume_cm3	Price_kes
0	Dettol Antiseptic Disinfectant	500	...	550	1000
1	Dettol Antiseptic Disinfectant	50	...	55	100
2	Carex Antiseptic Original	100	...	110	130
3	Savlon Antiseptic	250	...	275	335

```
4  Astonish Anti-Bacterial Cleanser        750  ...        800        445

[5 rows x 6 columns]
```

[ ]:
```
# need the columns size_in_kg and price_kes for the problem
weight_kg = pigeon_data['Weight_in_Kg']
volume_cm3 = pigeon_data['Volume_cm3']
price_kes = pigeon_data['Price_kes']
```

6.4 Experiment 1

[ ]:
```
# The data for the problem
# C_w is total weight capacity of a grocery delivery bike (Kg)
# C_v is the max volume capacity of the courier bag (cubic centimeters)
# weights,volumes and prices of each product are specified from the dataset␣
 ↪linked in the appendix
C_w = 9
C_v = 48000

#Setting the variables
weights = np.array(weight_kg)
volume = np.array(volume_cm3)
prices = np.array(price_kes)

# The variable we are solving for is x which assummes a boolean value of 1 or 0␣
 ↪indicating which products to select from the list
x = cp.Variable(len(pigeon_data['Brand']),boolean=True)

# The sum of the weights should be less than or equal to C_w = 9 Kg
weight_constraint = weights * x <= C_w

# The sum of the volume should be less than or equal to C_v = 48,000 Cubic␣
 ↪centimeters
volume_constraint = volume * x <= C_v

# Our basket value is the sum of the item prices
basket_value = prices * x

# We tell cvxpy that we want to maximize the total basket value subject to the␣
 ↪weight constraint and volume constraint.
# cvxpy must be passed as a list
const = [weight_constraint, volume_constraint]
knapsack_problem = cp.Problem(cp.Maximize(basket_value), const)

# Solving the problem
# The problem is solved using GLPK_MI which is a mixed integer LP solver in cvxpy
```

```
knapsack_problem.solve(solver=cp.GLPK_MI)


#Printing out the products selected
print ('Products selected:', x.value)
print ('Number of selected products:', len(x.value[x.value==1]))
print ('Objective value:', knapsack_problem.value )
```

```
Products selected: [1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
Number of selected products: 31
Objective value: 9820.0
```

6.5 Experiment 2

[ ]:
```
# The data for the problem
# Adjusting C_w to allow for 2 Delivery riders to service one grocery order
# C_v is the max volume capacity of the courier bag (cubic centimeters)
# weights,volumes and prices of each product are specified from the dataset␣
 ↪linked in the appendix
C_w = 18
C_v = 48000

#Setting the variables
weights = np.array(weight_kg)
volume = np.array(volume_cm3)
prices = np.array(price_kes)

# The variable we are solving for is x which assummes a boolean value of 1 or 0␣
 ↪indicating which products to select from the list
x = cp.Variable(len(pigeon_data['Brand']),boolean=True)

# The sum of the weights should be less than or equal to C_w = 9 Kg
weight_constraint = weights * x <= C_w

# The sum of the volume should be less than or equal to C_v = 48,000 Cubic␣
 ↪centimeters
volume_constraint = volume * x <= C_v

# Our basket value is the sum of the item prices
basket_value = prices * x

# We tell cvxpy that we want to maximize the total basket value subject to the␣
 ↪weight constraint and volume constraint.
# cvxpy must be passed as a list
```

```
const = [weight_constraint, volume_constraint]
knapsack_problem = cp.Problem(cp.Maximize(basket_value), const)

# Solving the problem
# The problem is solved using GLPK_MI which is a mixed integer LP solver in cvxpy
knapsack_problem.solve(solver=cp.GLPK_MI)


#Printing out the products selected
print ('Products selected:', x.value)
print ('Number of selected products:', len(x.value[x.value==1]))
print ('Objective value:', knapsack_problem.value )
```

```
Products selected: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0.
0. 0. 0. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0.]
Number of selected products: 46
Objective value: 13406.0
```

6.6 Experiment 3

[ ]:
```
# The data for the problem
# C_w is total weight capacity of a grocery delivery bike (Kg)
# weights and prices of each product are specified from the dataset linked in␣
 ↪the appendix
C_w = 9

#Setting the variables
weights = np.array(weight_kg)
prices = np.array(price_kes)

# The variable we are solving for is x which assummes a boolean value of 1 or 0␣
 ↪indicating which products to select from the list
x = cp.Variable(len(pigeon_data['Brand']),boolean=True)

# The sum of the weights should be less than or equal to C_w = 9 Kg
weight_constraint = weights * x <= C_w

# Our basket value is the sum of the item prices
basket_value = prices * x

# We tell cvxpy that we want to maximize the total basket value subject to the␣
 ↪weight constraint.

knapsack_problem = cp.Problem(cp.Maximize(basket_value), [weight_constraint])
```

```
# Solving the problem
# The problem is solved using GLPK_MI which is a mixed integer LP solver in cvxpy
knapsack_problem.solve(solver=cp.GLPK_MI)


#Printing out the products selected
print ('Products selected:', x.value)
print ('Number of selected products:', len(x.value[x.value==1]))
print ('Objective value:', knapsack_problem.value )
```

```
Products selected: [1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
Number of selected products: 31
Objective value: 9820.0
```