Final Project

A183 SQIT5013 KUMP A BUSINESS PROGRAMMING USING VISUAL TOOLS

Student Name:	Wong Jerng Foong
Metric Number:	824481
Lecturer:	Dr. Ruzelan Bin Khalid
Date:	September 2, 2019

List of Figures

Contents

1 INTRODUCTION

1.1 Introduction

The 0–1 Multidimensional Knapsack Problem (MKP) is one of the most well-known constrained integer programming problems that have received wide attention from the operational research community during the last four decades(Ktari & Chabchoub, 2013). Several names have been mentioned in the literature for the MKP: m-dimensional knapsack problem, multidimensional knapsack problem, multiknapsack problem, multiconstraint 0–1 knapsack problem, etc.

The multidimensional knapsack problem (MKP) can be stated informally as the problem of packing items into a knapsack while subject to the multiple constraints. The constraints can be, for example, maximum weight to be carried, the maximum available volume, and the maximum budget that can be afforded for the items in the problem. The goal is to select a combination of items that maximises their profits. The MKP problem is generally solved by approximate optimisation methods such as metaheuristics as compared to exact methods.

MKP is widely used in many areas such as: multi-unit combinatorial auctions(Pfeiffer & Rothlauf, 2007), allocation resources with stochastic demands (Luedtke & Ahmed, 2008), frequency allocation in cognitive radio networks (Mitola & Maguire, 1999), MP-Soc runtime management problem (Ykman-Couvreur, Nollet, Catthoor, & Corporaal, 2006), capital budgeting problem (Khalili-Damghani & Taghavifard, 2011), and real estate property maintenance (Taillandier, Fernandez, & Ndiaye, 2017)

In the past four decades, amongst the types of metaheurisitics used by researchers to solve MKP were Variable neighborhood search (VNS) (Hansen, Mladenović, Brimberg, & ..., 2019), Evolutionary Algorithm (EA) (Plata-González, Amaya, Ortiz-Bayliss, & ..., 2019; Sato & Ohnishi, 2018; Ferjani & Liouane, 2017), Genetic Algorithm(GA)(Rezoug, Bader-El-Den, & Boughaci, 2019; Kieffer, Danoy, Brust, & ..., 2019; Rezoug, Bader-El-Den, & Boughaci, 2018; Ohnishi & Ahn, 2017), Artificial algae algorithm (AAA)(Korkmaz, Babalik, & Kiran, 2018), Ant colony algorithm (ACA) (Montero, 2018; Morales, 2018; Mahrueyan, 2017) and Simulated Annealing(SA) (de Almeida Dantas & Caceres, 2018).

Metaheuristic methods have various advantages such as:

1. They are robust and can adapt solutions with changing conditions and environment

2. They can be applied in solving complex multimodal problems.

3. They may incorporate mechanisms to avoid getting trapped in local optima.

4. They are not problem -specific algorithm

5. These algorithms are able to find promising regions in a reasonable time due to exploration and exploitation ability.

6. They can be easily employed in parallel processing

My aim in this study is to design a metaheuristics based on tabu search to solve the MKP problem using python as the coding framework.

1.2 Research Question

The research question that we propose for this study are:

1. What are the suitable techniques to construct initial solution?

2. How to improve the initial solution?

3. How to evaluate the quality of the proposed techniques?

1.3 Research Objectives

The research objectives are:

1. To construct initial solution using greedy heuristic

2. To improve the initial solution using TS

3. To evaluate the quality of the proposed techniques using the MKP benchmark

1.4 Scope of Study

To achieve the mentioned objectives, the scope of this study is bounded as follows:

1. mknapcb7.txt dataset for MKP from OR-Library:(http://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html). The data files are the problems solved in (Chu & Beasley, 1998).

2. The metaheuristic technique used is Tabu Search only.

3. The programs have been customized, developed and applied to the problems using Python 3.7 language.

2 METHODOLOGY

2.1 Problem Analysis

Input: Data files from OR library named mknapcb7.txt.

The data files are the problems solved in P.C.Chu and J.E.Beasley "A genetic algorithm for the multidimensional knapsack problem", Journal of Heuristics, vol. 4, 1998, pp63-86.

The problem to be solved is:

$$Maximise, Z = \sum_{j=1}^{n} p_j x_j$$

$$s.t. \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \quad i = 1, \ldots, m,$$

$$x \in \{0, 1\} \quad j = \{1, \ldots, n\}$$

The format of this data file(See Fig. 1) is:

- number of test problems (K)
- then for each test problem $k(k = 1, ..., K)$ in turn:

  number of variables (n), number of constraints (m), optimal solution value (zero if unavailable)
- the coefficients $p(j); j = 1, ..., n$
- for each constraint $i(i = 1, ..., m)$:

  the coefficients $r(i, j); j = 1, ..., n$

  the constraint right-hand sides $b(i); i = 1, ..., m$

Processing: After that it needs to be processed by the TS algorithm (Flowchart 2.3). The algorithm would be initialisation, neighbouring, tabu list maintenance, scoring and selection

Output:The output from the program would be a text file containing the best solution in binary form(Fig. 2), and a graphical display of the solution (Fig. 3)

2.2 Tabu Search Algorithm Flowchart

For the flowchart, please refer to Figure 4. Below is the notation used.

mknapcb7.txt

```
30
100 30 0
1002 889 774 746 873 916 731
660 601 792 647 623 566 574
953 922 927 670 814 661 590
1013 640 911 650 670 1067 988
765 955 857 943 502 584 761
599 507 804 462 558 648 484
898 1052 568 617 690 665 754
604 564 498 804 496 1013 834
561 873 859 578 534 819 619
953 497 843 800 934 757 570
946 691 1006 1027 714 630 565
955 828 564 601 631 823 657
617 674 624 647 506 1020 870
929 921 484 867 926 748 583
882 586
193 665 929 958 962 186 666
630 615 67 334 823 341 590
734 747 992 826 276 645 572
127 673 587 322 846 792 437
180 774 675 827 287 318 705
416 619 494 468 720 276 532
148 186 349 631 598 549 521
365 618 4 906 725 542 994
702 676 766 540 317 88 505
760 275 206 874 481 213 283
795 688 957 405 377 338 981
583 576 956 173 143 381 927
669 256 981 67 274 486 600
```

Figure 1: Screenshot of data

solution.txt

```
13196,16727,[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
15322,18192,[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
13196,16748,[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
15322,18384,[1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
14739,18639,[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0]
```

Figure 2: Output in text file

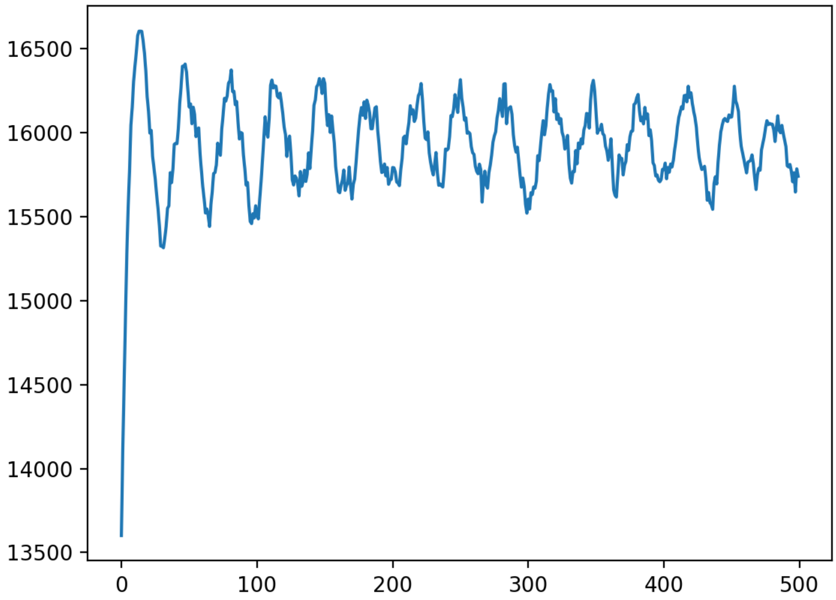

Figure 3: Solution over iterations run

Notation used for Flowchart

- ***S*** , the current solution,
- ***S'*** , the current best solution in current Neighbourhood N(S),
- ***S**** , the best-known solution,
- ***f**** , value of $\mathbf{S}^*$,
- $\boldsymbol{N(S)}$, the neighbourhood of $\mathbf{S}^*$,
- ***T*** , Tabu list

2.3 Program Design

The program is divided into two main files: `tabunested.py` and `setsnested.py`. The main algorithm coding is placed at `tabunested.py` . `setsnested.py` contains instructions for data loading and final execution.

The two files are separated for better scaleability and reusability. For example, if another format of datasets is required, we would just need to amend `setsnested.py` only without changing the main algorithm program.

2.3.1 Reading Data File

The first step towards running the optimisation program is reading the data file. The main function performing this function is `loader()` in `setsnested.py` and consists of two nested functions, `readlist()` and `readsets()`.

The other objective of the `loader()`function is to break the file into datasets. Each text file consists of 30 datasets, without any clear separators. The program needs to know the position where each datasets begins and ends. When run, `loader()` outputs a list of 30 segregated lists(since there are 30 sets of datasets in each mknap.txt file).

`readlist()` converts string into a list. `open()` and `read()` reads the lines into memory as string. `tolist()` converts the string into an array.

```
def readlist():
        f=open("/Volumes/WINDATA/Thesis/dissertation1/data/mknapcb7.txt")
        string1=(f.read())
        list1=tolist(string1)

        return list1
```

In `tolist()`, `string.strip()` and `string.rstrip()` returns a copy of the string with both leading and trailing characters, and blank spaces removed. `string.split()` returns a list of strings after breaking the given string by the specified separator. `list.append()` appends value in brackets to a list.

```
def tolist(string):
        string0=string.strip()
        string1 = string0.rstrip('\n')
        list1=string1.split(" ") # this converts the line number text to a list

        list2=[]
        for x in list1:# this removes blanks and puts into new list
```

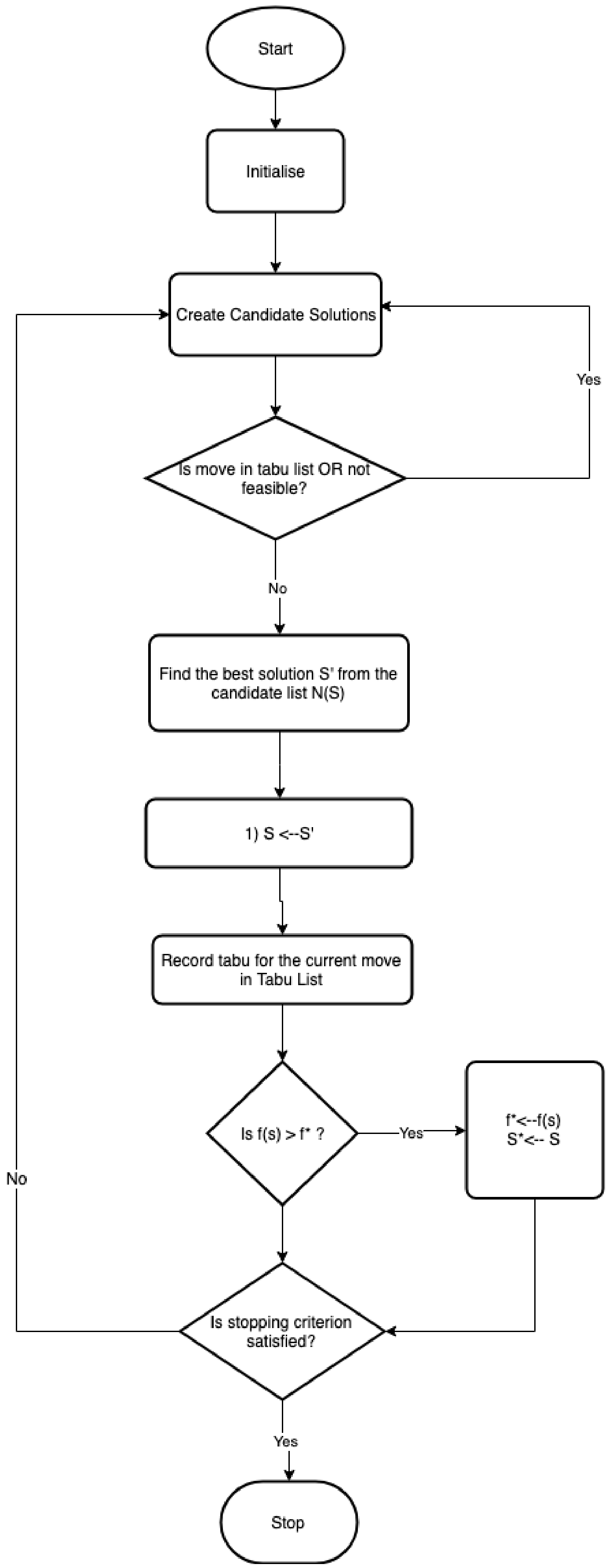

Figure 4: Tabu Search Algorithm Flowchart

```
            if x !="":
                if x!='\n':
                    list2.append(int(x))
        return list2
```

Next, `readsets()` will size up each sets according to its number of integer determined by `size()` into a list of lists.

```
    def readsets(list1):
        start=1
        list2=[]
        for i in  range(30):
            v=list1[start]
            c=list1[start+1]
            z=size(v,c)
            list2.append(list1[start:start+z])
            start = start+z
        return list2
```

`size()` is to determine the size of 1 set and determine starting point for function to append integer values into a new list.

```
def size(v,c):
        "determine size of 1 set"
        setsize=3+v+(v*c)+c
        return setsize
```

2.3.2 Processing Metaheuristic Algorithm

In this section, is set main metaheuristic algorithm for finding the solution using optimisation strategy. The main function is `main()` and ties all the functions together.

The individual functions will be discussed in detailed in the subsections.

```
def main(iterations):
        #global tt #values of tt updated in function updates global
        s=initialise(variable)
        seed,tt,initialfitness =s[0],s[1],s[2]
        bestfitness=0
        best=()
        bestlist=[]
```

```
        iterationlist=[]
        for k in range(iterations):
            #reducetabu(tt) #each iteration reduce tabu tenure
            list1=candidatelist(seed,tt) # generate candidates
            reducetabu(tt)
            temp=bestcandidate(list1)
            if temp[1]!=0: # to guard against 0 value
                solution=temp
            if solution[1]>bestfitness:
                bestfitness=solution[1]#assign best int value
                best=solution[:]#clone the list binary solution to best list
            seed=solution[0][0]# clone binary to seed for next iteration
            updatetabu(solution,tt,variable)# update tabu moves
            bestlist.append(solution[1])#this is used to plot iteration axis in graph
            iterationlist.append(k)#this is used to plot iteration axis in graph
            #print("S",solution)
            #print("B","k",k,best,)
            f=fitness(solution[0][0])
        y,x=bestlist,iterationlist
        plt.plot(x,y) #uncomment to show graph
        #plt.plot(best[1],k)
        plt.show() #uncomment to show graph
        return best,y,x,initialfitness
```

2.3.3 Initialisation

In the initialisation phase, an attempt is made to construct a feasible solution to be the initial solution for further improvement in the neighbouring phase.

In the beginning, all binary values in the gene at set to zero. Then step-wise, beginning with gene at first position, the first gene is added as '1' value and feasibility is checked against all constraints. If feasible, the process will continue to next gene, where it is flipped to '1' value. Process is repeated until value of solution exceeds constraints, and then the last feasible solution is used as initial solution.

```
def initialise(variable):#x in length of variable
        initial=[]
        for i in range(variable):
```

```
            initial.append(0)
        tt=initial[:]
        #create list of zeros of length x
        for j in range(len(initial)):
            initial[j]=1
            if isfeasible_mult_constraints(constraints,initial):
                continue
            else:
                initial[j]=0 #reset back to zero
                initialfitness = fitness(initial)
                return initial,tt,initialfitness
```

isfeasible_mult_constraints() tests for feasibility and that the solution meets all constraints. fitness() scores the solution and determines objective value. initialise() returns initial, a list, tt -a copy of the initial list, and inititialfitness -an integer value,which is the fitness score of the initial solution.

2.3.4 Neighbouring

Local Search move is done by one 'Add' move plus one 'Drop' move. The determination of which values to add/drop are random . Add and Drop moves are constrained to moves that are not found in tabu list.

candidatelist() will generate a list of candidates. Each 'candidate' is a list of binary values, with a feasible move implemented.move() generates tuple ,where first value of tuple is gene, and next tuple the add, drop tabu positions. Number of candidates are the length of the gene.

```
    def candidatelist(list1,tt):
        list2=[]
        if isinstance(list1, tuple):
            for i in range(len(list1[0])):
                list2.append(move(list1[0][0],tt))
        else:
            for i in range(len(list1)):
                list2.append(move(list1,tt))
        return list2
```

move() consists of one add() move and one drop() move.

```
    def move(list1,tt):
        list3=[]
        finallist=[]
```

```
    list3,droptabu=add(list1,tt)
    finallist,addtabu=drop(list3,droptabu,tt)
    return finallist,droptabu,addtabu

```

In `add()` , `indexpositions()` identifies genes index positions which are of value "0" in a list, and therefore eligible for flipping to "1". `random.choice()` makes a random pick . If the pick is found to be in tabu list via `istabu()`, another pick is done till eligible candidate gene is found. `flip()` will flip the "0" genes selected to "1" value. The function returns a list(`list3`), which has been modified with the move, plus an integer (`pick`) value signifying the index position where the add move has been made. This is used later to update the tabu list.

```
def add(list1,tt):
    list2=[] #list to store "0" indexes
    list3=[] #new return list
    list2=indexpositions(list1,0)#list of "0" indexes
    pick=random.choice(list2) #pick a initialchoice of the index no."Pick" is int
    while istabu(tt,pick): #while pick is tabu,repeat till valid pick is found
        pick=random.choice(list2)
    list3=flip(pick,list1)
    return list3,pick

```

`drop()` works in the same way accept in reverse, where it identifies '1' values, and flips selected ones to '0'. The function also enforces tabu rules and restricts the pick from selecting the same 'add' move.

```
def drop(list1,addpick,tt):
    list2=[] #list to store "0" indexes
    list3=[] #new return list
    list2=indexpositions(list1,1)#list of "1" indexes
    pick=random.choice(list2) #pick a initialchoice of the index no."Pick" is int
    while istabu(tt,pick) or pick==addpick: #while pick is tabu,repeat till valid pick is found
        pick=random.choice(list2)
    list3=flip(pick,list1)
    return list3,pick

```

2.3.5 Reduction of Tabu

The next step is then the Reduction of tabu process. This process takes place during each iteration, and all values in tabu list which is non-zero is reduced by one. This is performed by the `reducetabu()` unto the tabu list (`tt`). In `main()`, the tabu tenure is reduced by 1, during every iteration cycle. Using a `for` loop, every element in `tt` is checked for non-zero values, and if so, the element is reduced by value of 1.

```
    def reducetabu(tt):
        for i in range(len(tt)):
            if tt[i]!=0:
                tt[i]=tt[i]-1

```

2.3.6 Scoring and Selection

Next, scoring process is done whereby the individual candidates are evaluated its fitness with function `bestcandidate()`.

Each candidate in list is evaluated with `fitness()` function. The best candidate binary solution, and its fitness value is assigned to `besttuple` and `besttuplefitness` variable respectively. The candidate must fulfil feasibility and meet constraint requirements.This is validated via `isfeasible_mult_constraints()`. The function returns the best solution, and its fitness score.

```
def bestcandidate(list1):
        besttuple=()
        besttuplefitness=0
        for t in list1:
            f=fitness(t[0])
            if f>besttuplefitness and isfeasible_mult_constraints(constraints,t[0]):
                besttuple=t
                besttuplefitness=f
        return besttuple,besttuplefitness

```

The best solution value is assigned to `bestfitness`. The best solution binary is assigned to `best`. `[:]` method clones a copy of a list to new list. It is a deep copy function.

Current best solution is saved to `seed` variable, to be used for next iteration.

```

if solution[1]>bestfitness:
```

```
  bestfitness=solution[1]#assign best int value
  best=solution[:]#clone the list binary solution to best list
seed=solution[0][0]# clone binary to seed for next iteration

```

2.3.7 Update tabu list

The tabu list is updated with the current solution moves via `updatetabu()` Input is values from `move()`.

```
    def updatetabu(tuple1,tt,variable):
        droptenure = 15
        addtenure = 15
        tt[tuple1[0][1]]=droptenure
        tt[tuple1[0][2]]=addtenure

```

2.3.8 Graph plotting

`bestlist` and `iterationlist` are variables used to plot graph using `.plot()` and `.show()` method from the `matplotlib` module.

```
bestlist.append(solution[1])#this is used to plot iteration axis in graph
iterationlist.append(k)#this is used to plot iteration axis in graph
plt.plot(x,y)
plt.show()

```

2.3.9 Stopping Criterion

The stopping criterion for the algorithm is set by the `n`(number of iterations) parameter in `main`.

2.3.10 File output

The variable f is declared to open a file named `solution.txt`. `Open()` takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file. Here we used `"a"` letter in our argument, which indicates append and the `+` sign that means it will create a file if it does not exist in library. `print()` displays output on the screen. `write()` is used to enter data into the file. `close()` will close the instance of the file `solution.txt` stored.

```
l2=loader()#returns list of list
counter=0
for i in l2:#i is for one list
    a,b=(counter,engine(500,i))
    print(a,b)
    f= open("solution.txt","a+")
    f.write("%d,%d,%s\r\n" % (b[0],b[1],b[2]))
    f.close()
    counter=counter+1

```

2.3.11 Program execution

tabunested.py and setsnested.py file has to be placed in the same directory. Program is executed by running setsnested.py. The import command will import relevant modules (random, matplotlib.pyplot, tabunested) into the current instance required necessary to start the algorithm.

```
import random
import matplotlib.pyplot as plt
from tabunested import engine

```

3 RESULT

Results were compared to 5 benchmark results from (Chu & Beasley, 1998), and tabled below. It is found that the performance of the algorithm was fairly good though not fully reaching the benchmark results. It is noted that the method used in (Chu & Beasley, 1998) was a hybrid between Genetic Algorithm and Tabu search.

Instance	ChuBeasley	Wong	Comparison
30.100-25	60011	57060	0.95
30.100-26	58025	55826	0.96
30.100-27	60776	57290	0.94
30.100-28	58884	55570	0.94
30.100-29	60603	58809	0.97

4 CONCLUSION

This project has successfully designed a program using Tabu Search metaheuristic to find a good solution to the MKP problem in Python language. Although the algorithm did not fully reach or exceed the original benchmark, it was very near. To further improve the result, further studies could perhaps be performed to utilise hybrid metaheuristics, as this has shown promising results in recent times.

5 CODE

This section lists the coding used in the project.

5.1 tabunested.py

```
import random

import matplotlib.pyplot as plt


def engine(n,list1):


    def listoflist(original_list,variable):

        "converts list of int to list of lists"

        start=0

        list1=[]

        for i in range(int(len(original_list)/variable)):

            list1.append(original_list[start:start+variable])

            start = start + variable

        return list1


    def listopen(list1):

        """

        input:list

        returns tuple of 2 int and 3 lists

        ######

        Parameter 'list1' = a list of single int passed from loader()


        Purpose: The purpose of this function to 'divide' the components of

        list1 into the respective sections, for further processing.The sections

        are variable, constraint, optimal value(0 if none), list of coefficients,

        list of constraints, and list of Right Hand Side(RHS) values.
```

```
    listoflist() is to convert a list into a list of lists. Constraints
    in its proper form is a list of constraint coefficients , by number of
    constraints.
    """
    variable=list1[0]
    constraint=list1[1]
    optimal = list1[2]
    initial=3
    list_coefficient=list1[initial:initial+variable]
    list_constraint = list1[initial+variable:-constraint]
    newlist=listoflist(list_constraint,variable)
    list_rhs=list1[-constraint:len(list1)]

    return variable ,list_coefficient,newlist,list_rhs,optimal




'''
def constraintlist(r):
    """
    returns list of list
    This was important, because it helped me to learn how
    to make a list of lists from a list. Basically every iteration
    started a new list.

    """
    list3=[]
    for i in r:
```

```
        a=tolist(i)
        list3.append(a)
    return list3
'''

def fitness(listgene):
    """
    returns int
    -----------------------------------
    Parameter 'listgene' = List of binaries(0,1)

    Purpose: This returns an int representing the fitness of a
            binary solution

    Variable 'coeff' is a variable passed from listopen()

    """
    valuelist=[]
    for i in range(len(listgene)):
        score=coeff[i]*listgene[i]
        valuelist.append(score)
    v=sum(valuelist)
    return v

def isfeasible_constraint(constraint,rhs,genes):
    """
    returns True
    -------------------------------------------

    Parameter 'constraint' = list of constraints
```

```
    Parameter ' rhs' = list of RHS values
    Parameter 'genes' = list of int values, representing binary
    All these parameters are returned from listopen()

    Purpose: Returns true when all constraints are satisfied i.e

    LHS one constraint <= RHS

    """
    list2=[]
    for i in range(len(constraint)): # boolean check for single list
        s1=constraint[i]*genes[i]
        list2.append(s1)

    if sum(list2)<=rhs:
        return True
    else:
        return False

def isfeasible_mult_constraints(constraints,list1):
    """
    returns Boolean

    Parameter 'constraints' from fileopen()
    Parameter 'list1' = 1 list of gene values
    """
    for j in range(len(constraints)): # selects a list from lists
        if isfeasible_constraint(constraints[j],rhs[j],list1):
            continue
        else:
```

```
            return False
    return True



def indexpositions(list1,int1):
    """
    to return positions of int in list

    Parameter 'list1' would be list of int
    Parameter 'int1' would be either '0' or '1'

    """
    list2=[]
    for i in range(len(list1)):
        if int1==list1[i]:
            list2.append(i)
    return list2

def istabu(list1,int1):
    """
    returns Boolean
    ----------------------------------------
    Parameter 'list1' is tabu list
    Parameter 'int' is index ,which refers to the index of the tabu ist
    """
    if list1[int1]>0:
        return True

def move(list1,tt):
    """
```

```
    return one tuple (list,int,int)

    ----------------------------------------
    Parameter 'list1' would be the candidate gene list

    Purpose: This generates a move candidate, with the
    index of the add,drop tabu move. If this candidate is
    selected, the add/drop tabu move will update the tabu list

    """
    list3=[]
    finallist=[]
    list3,droptabu=add(list1,tt)
    finallist,addtabu=drop(list3,droptabu,tt)
    return finallist,droptabu,addtabu


def updatetabu(tuple1,tt,variable):
    """
    input:solution tuple
    updates tt list
    -------------------------------------
    Parameter 'tuple1' = tuple of the best candidate, with drop/add tabu moves

    Variable tt, representing tabu list, is declared at line 315

    """
    #global tt
    #droptenure = int(variable/25)
    #addtenure = int(variable/50)
```

```
    droptenure = 15
    addtenure = 15
    tt[tuple1[0][1]]=droptenure
    tt[tuple1[0][2]]=addtenure
    #return finallist,droptabu,addtabu



def flip(index,list1):
    """
    flip a binary value
        returns list

    Parameter 'index' is an integer given by 'pick' variable in the
    add() or drop() function.

    """
    list2=list1[:] #clones a list
    if list2[index]==0:
        list2[index]=1
    else:
        list2[index]=0
    return list2



def add(list1,tt):
    """
    returns tuple, of a list, and str
    -----------------------------------
    Parameter 'list1' = candidate list
```

```
    """
    list2=[] #list to store "0" indexes
    list3=[] #new return list
    list2=indexpositions(list1,0)#list of "0" indexes
    pick=random.choice(list2) #pick a initialchoice of the index no."Pick" is
     ↪ int
    while istabu(tt,pick): #while pick is tabu,repeat till valid pick is found
        pick=random.choice(list2)
    list3=flip(pick,list1)
    return list3,pick

def drop(list1,addpick,tt):
    """returns a list, and tabu list index"""
    list2=[] #list to store "0" indexes
    list3=[] #new return list
    list2=indexpositions(list1,1)#list of "1" indexes
    pick=random.choice(list2) #pick a initialchoice of the index no."Pick" is
     ↪ int
    while istabu(tt,pick) or pick==addpick: #while pick is tabu,repeat till
     ↪ valid pick is found
        pick=random.choice(list2)
    list3=flip(pick,list1)
    return list3,pick

def reducetabu(tt):
    """
    updates tabu list

    Parameter 'tt' is a list
```

```
    """
    #global tt
    for i in range(len(tt)):
        if tt[i]!=0:
            tt[i]=tt[i]-1

def candidatelist(list1,tt):
    """
    returns list of tuples

    Parameter 'list1' is the starting gene to generate multiple candidates lists

    move() generates tuple ,where first value of tuple is gene, and next tuple
    the add, drop tabu positions.

    Number of candidates are the length of the gene.

    """
    list2=[]
    if isinstance(list1, tuple):
        for i in range(len(list1[0])):
            list2.append(move(list1[0][0],tt))
    else:
        for i in range(len(list1)):
            list2.append(move(list1,tt))
    return list2



def bestcandidate(list1):
    """
```

```
    input:list of tuples

    evaluates list of tuples and

    returns: tuple

    """

    besttuple=()

    besttuplefitness=0

    for t in list1:

        """

        if len(t)==2:

            f=fitness(t[0][0])

        else:

            f=fitness(t[0])

        """

        f=fitness(t[0])

        if f>besttuplefitness and isfeasible_mult_constraints(constraints,t[0]):

            besttuple=t

            besttuplefitness=f

    return besttuple,besttuplefitness


def initialise(variable):#x in length of variable

    """input:int

        returns: list



    Parameter 'variable' is an int, which represent the number of x values.


    Tabu list, tt, is cloned from initial list,which is a list of zeros.
```

```
    """
    initial=[]

    for i in range(variable):
        initial.append(0)
    tt=initial[:]
    #create list of zeros of length x
    for j in range(len(initial)):
        initial[j]=1
        if isfeasible_mult_constraints(constraints,initial):
            continue
        else:
            initial[j]=0 #reset back to zero
            initialfitness = fitness(initial)
            return initial,tt,initialfitness



def main(iterations):
    """
    Parameter 'iterations' is the number of times improvement is run

    """
    #global tt #values of tt updated in function updates global
    s=initialise(variable)

    seed,tt,initialfitness =s[0],s[1],s[2]
    bestfitness=0
    best=()
    bestlist=[]
    iterationlist=[]
```

```
    for k in range(iterations):
        #reducetabu(tt) #each iteration reduce tabu tenure
        list1=candidatelist(seed,tt) # generate candidates
        reducetabu(tt)
        temp=bestcandidate(list1)
        if temp[1]!=0: # to guard against 0 value
            solution=temp
        if solution[1]>bestfitness:
            bestfitness=solution[1]#assign best int value
            best=solution[:]#clone the list binary solution to best list
        seed=solution[0][0]# clone binary to seed for next iteration
        updatetabu(solution,tt,variable)# update tabu moves
        bestlist.append(solution[1])#this is used to plot iteration axis in
         ↪ graph
        iterationlist.append(k)#this is used to plot iteration axis in graph
        #print("S",solution)
        #print("B","k",k,best,)
        f=fitness(solution[0][0])
    y,x=bestlist,iterationlist
    plt.plot(x,y) #uncomment to show graph
    #plt.plot(best[1],k)
    plt.show() #uncomment to show graph
    return best,y,x,initialfitness


#########INITIALISATION#############
#this statement to convert tuple return values from listopen()
#to variables in engine()
variable,coeff,constraints,rhs,optimal=listopen(list1)
```

```
    #initialise tabu list, tt
    #tt = initialise(variable)[1]

    #return values from main()
    e=main(n)
    #return tt
    return e[3],e[0][1],e[0][0][0]
```

5.2 setsnested.py

```
def loader():

    def readlist():
        """
        returns :list of int
        tolist parses the list to proper integers
        Reads text file and returns a list

        """
        f=open("/Volumes/WINDATA/Thesis/dissertation1/data/mknapcb7.txt")
        string1=(f.read())
        list1=tolist(string1)

        return list1

    def readsets(list1):
        '''
        returns list of lists

        start from 1 , because position 0 is the set number.
        size() is to determine the size of 1 set
```

```
    This loop will size up each sets according to its number of int
    determined by size() into a list of lists
    v=variable in text file
    c = contraint in text file
    '''
    start=1
    list2=[]
    for i in  range(30):
        v=list1[start]
        c=list1[start+1]
        z=size(v,c)
        list2.append(list1[start:start+z])
        start = start+z
    return list2


def size(v,c):
    "determine size of 1 set"
    setsize=3+v+(v*c)+c
    return setsize

def tolist(string):
    """
    returns list
    Parse string to two levels. First to remove space, then \n
    """
    string0=string.strip()
    string1 = string0.rstrip('\n')
    list1=string1.split(" ") # this converts the line number text to a list
```

```
        list2=[]
        for x in list1:# this removes blanks and puts into new list
            if x !="":
                if x!='\n':
                    list2.append(int(x))
        return list2

    ll=readsets(readlist())
    return ll # the main output of this nested function

####RUN##########################################################################
import random
import matplotlib.pyplot as plt
#from setsnested import loader
from tabunested import engine

l2=loader()#returns list of list
counter=0
for i in l2:#i is for one list
    a,b=(counter,engine(500,i))
    print(a,b)
    #print(counter,engine(500,i))
    f= open("solution.txt","a+")
    f.write("%d,%d,%s\r\n" % (b[0],b[1],b[2]))
    f.close()
    counter=counter+1
```

References

Chu, P. C., & Beasley, J. E. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Springer Journal of Heuristics*, *4*(1), 63–86. Retrieved from https://link.springer.com/article/10.1023/A:1009642405419

de Almeida Dantas, B., & Caceres, E. N. (2018). An experimental evaluation of a parallel simulated annealing approach for the 0–1 multidimensional knapsack problem. *Journal of Parallel and Distributed* .... Retrieved from https://www.sciencedirect.com/science/article/pii/S0743731518301096

Ferjani, A. A., & Liouane, N. (2017). Logic gate-based evolutionary algorithm for the multidimensional knapsack problem. *2017 International Conference on* .... Retrieved from https://ieeexplore.ieee.org/abstract/document/8075650/

Hansen, P., Mladenović, N., Brimberg, J., & ... (2019). Variable neighborhood search. *Handbook of* .... Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-91086-4_3

Khalili-Damghani, A., & Taghavifard, M. (2011). Solving a bi-objective project capital budgeting problem using a fuzzy multi-dimensional knapsack..

Kieffer, E., Danoy, G., Brust, M. R., & ... (2019). Tackling Large-Scale and Combinatorial Bi-level Problems with a Genetic Programming Hyper-heuristic. *IEEE Transactions on* .... Retrieved from https://ieeexplore.ieee.org/abstract/document/8671761/

Korkmaz, S., Babalik, A., & Kiran, M. S. (2018). An artificial algae algorithm for solving binary optimization problems. *International Journal of Machine* .... Retrieved from https://link.springer.com/article/10.1007/s13042-017-0772-7

Ktari, R., & Chabchoub, H. (2013). Essential particle swarm optimization queen with tabu search for mkp resolution. *Computing*, *95*(9), 897–921.

Luedtke, J., & Ahmed, S. (2008). A Sample Approximation Approach for Optimization with Probabilistic Constraints. *SIAM Journal on Optimization*. doi: 10.1137/070702928

Mahrueyan, M. (2017). *A proposal for an improved version of EigenAnt algorithm with performance evaluation on combinatorial optimization problems*. pantheon.ufrj.br. Retrieved from http://pantheon.ufrj.br/handle/11422/6239

Mitola, J., & Maguire, G. Q. (1999). Cognitive radio: making software radios more personal. *IEEE Personal Communications*, *6*(4), 13–18. doi: 10.1109/98.788210

Montero, E. (2018). A Cooperative Opposite-Inspired Learning Strategy for Ant-Based Algorithms. … : *11th International Conference, ANTS 2018, Rome* …. Retrieved from https://books.google.com/books?hl=en&lr=&id=hB9ODwAAQBAJ&oi=fnd&pg=PA316&dq=or-library+beas

Morales, N. E. R. (2018). *OPPOSITE LEARNING STRATEGIES FOR IMPROVING THE SEARCH PROCESS OF ANT-BASED ALGORITHMS*. repositorio.usm.cl. Retrieved from https://repositorio.usm.cl/handle/11673/42279

Ohnishi, K., & Ahn, C. W. (2017). Simple Linkage Identification Using Genetic Clustering. *Asia-Pacific Conference on Simulated Evolution and* …. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-68759-9_31

Pfeiffer, J., & Rothlauf, F. (2007). *Analysis of greedy heuristics and weight-coded eas for multidimensional knapsack problems and multi-unit combinatorial auctions*. doi: 10.1145/1276958.1277258

Plata-González, L. F., Amaya, I., Ortiz-Bayliss, J. C., & ... (2019). Evolutionary-based tailoring of synthetic instances for the Knapsack problem [HTML]. *Soft Computing*. Retrieved from https://link.springer.com/article/10.1007/s00500-019-03822-w

Rezoug, A., Bader-El-Den, M., & Boughaci, D. (2018). Guided genetic algorithm for the multidimensional knapsack problem. *Memetic Computing*. Retrieved from https://link.springer.com/article/10.1007/s12293-017-0232-7

Rezoug, A., Bader-El-Den, M., & Boughaci, D. (2019). Hybrid Genetic Algorithms to Solve the Multidimensional Knapsack Problem. *Bioinspired Heuristics for* …. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-95104-1_15

Sato, T., & Ohnishi, K. (2018). A Metaheuristic Relying on Random Walk on a Graph for Binary Optimization Problems. … *on Systems, Man, and Cybernetics (SMC)*. Retrieved from https://ieeexplore.ieee.org/abstract/document/8616130/

Taillandier, F., Fernandez, C., & Ndiaye, A. (2017, mar). Real Estate Property Maintenance Optimization Based on Multiobjective Multidimensional Knapsack Problem. *Computer-Aided Civil and Infrastructure Engineering*, *32*(3), 227–251. Retrieved from https://doi.org/10.1111/mice.12246 doi: 10.1111/mice.12246

Ykman-Couvreur, C., Nollet, V., Catthoor, F., & Corporaal, H. (2006). Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. *2006 International Symposium on System-on-Chip, SOC*, *10*(3), 1–16. doi: 10.1109/ISSOC.2006.321966