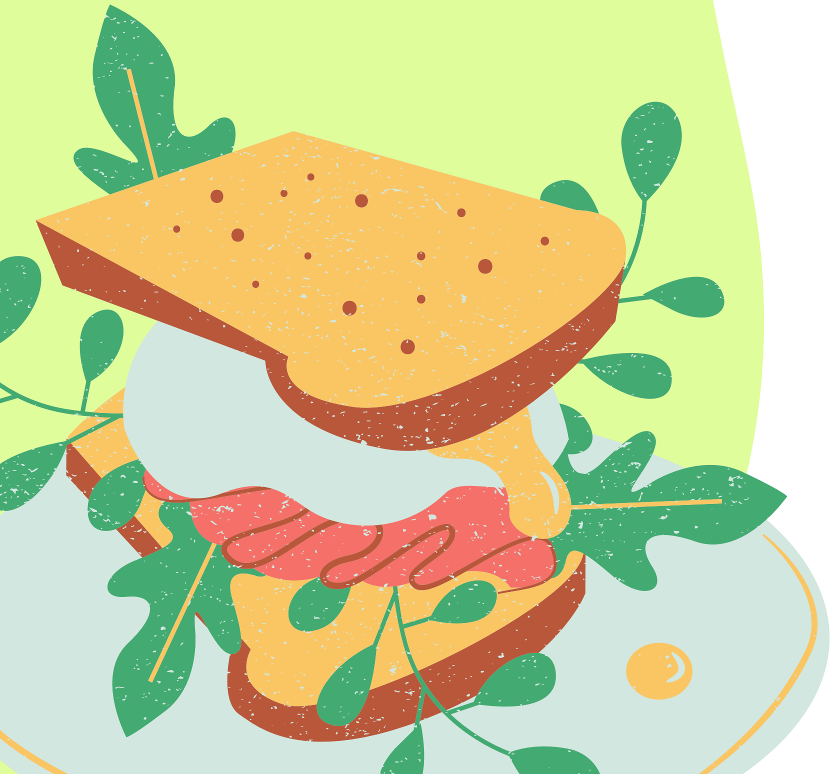
company lunch

CCAI-311: Optimization and Regression
PROJECT



Describe of the problem

A company is making lunch for a business meeting.
It will serve chicken sandwiches, light chicken sandwiches, and vegetarian sandwiches.
A chicken sandwich has 1 serving of vegetables, 4 slices of chicken, 1 slice of cheese, and 2 slices of bread. The price of each sandwich is 10 SR. A light chicken sandwich has 2 servings of vegetables, 2 slices of chicken, 1 slice of cheese, and 3 slices of bread, the price of each sandwich is 15 SR. A vegetarian sandwich has 3 servings of vegetables, 4 slices of cheese, and 3 slices of bread. The price of each sandwich is 20 SR.
In total, the company has available 5 bags of chicken, each of which has 80 slices of chicken, 10 loaves of bread, each with 30 slices, 200 servings of vegetables, and 9 bags of cheese, each with 100 slices of cheese.
How many of each sandwich can be produced if the goal is to maximize the benefit of sandwiches?

The goal is to maximize the benefit.

We wish to maximize the number of sandwiches, so let:

X= chicken sandwich
Y= light chicken sandwich
Z= vegetarian sandwich

The total price and cost of sandwiches are given by:

benefit=price - cost

The constraints

chicken sandwich, light chicken sandwich the required number of chicken
$4X+2Y<=400$
chicken sandwich, light chicken sandwich, vegetarian sandwich the required number of bread
$2X+3Y+3Z<=300$
chicken sandwich, light chicken sandwich, and vegetarian sandwich, with the required number of vegetables.
$X+2Y+3Z<=200$
chicken sandwich, light chicken sandwich, and vegetarian sandwich, with the required number of slices of cheese.
$X+Y+4Z<= 900$

formal representation

Maximum The Objective function:

$$p=10x+15y+20z$$

Subject to

$$4X+2Y<=400$$

$$2X+3Y+3Z<=300$$

$$X+2Y+3Z<=200$$

$$X+Y+4Z<= 900$$

$$X,Y,Z>=0$$

We used two metaheuristics to solve this problem:

- BASIC LOCAL SEARCH
- TABU SEARCH

BASIC LOCAL SEARCH

```python
# check_constraints & Objective function
def setList(x,y,z,p):
    '''
        f=10*X+15*Y+20*Z
        4X+2Y<=400
        2X+3Y+3Z<=300
        X+2Y+3Z<=200
        X+Y+4Z<= 900
        X,Y,Z>=0
    '''
    list=[ ]
    for i in range(0,len(x)):
        for j in range(0,len(y)):
            for l in range(0,len(z)):
                if x[i]>=0 and y[j]>=0 and z[l]>=0:
                    if 4*x[i]+2*y[j]<=400 and 2*x[i]+3*y[j]+3*z[l]<=300 and x[i]+2*y[j]+3*z[l]<=200 and x[i]+y[j]+4*z[l]<=900:
                        P=10*x[i]+15*y[j]+20*z[l]
                        list.append([x[i], y[j], z[l], P])
    return list
# Generate initial solution
def get_InitialSolution(list, setOfSolu):
    for i in range(0,len(setOfSolu)):
        if i == 0:
            optimalSolu = setOfSolu[0][3]
        else:
            optimalSolu = max(optimalSolu, setOfSolu[i][3])

        for i in range(0,len(setOfSolu)):
            if setOfSolu[i][3] == optimalSolu:
                initial_solution = setOfSolu[i]

        for i in range(0,len(list)):
            if list[i][3] == optimalSolu :
               index = i
    return initial_solution, index
#BasicLocalSearch
def BLS(list, index, initsolution):
    initial_solution=initsolution
    list1=[]
    list2=[]
    repeat=20
    optima =0
    # Genertae neighbour
    for i in range(repeat):
```

BASIC LOCAL SEARCH

```python
    for i in range(repeat):
        neighborhood1 = list[index + i]
        list1.append(neighborhood1)

        neighborhood2 = list[index - i]
        list2.append(neighborhood2)
    #look for improov
    for i in range(len(list1)):
        maax = max(initial_solution[3], list1[i][3], list2[i][3])
        if (maax > optima):
            optima = maax
        else:
            optima = optima
    #set improov
    for i in range(len(list)):
        if (list[i][3] == optima):
            initialsolution = list[i]
            index =i
    return index, initialsolution
#check_neighborhood
def neighborhood(index, solution, prvsolution):
    '''
        Takes a  solution
        returns a new neighbor solution
    '''
    if solution == prvsolution:
        return solution
    else:
        prvsolution = solution
        index, solution = BLS(list, index, solution)
        return neighborhood(index, solution, prvsolution)
x= [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
y= [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
z= [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
list=setList(x,y,z,0)
x= [4,2,1,1]
y= [2,3,2,1]
z= [0,3,3,4]
firstSolution, index1 = get_InitialSolution(list,setList(x, y, z, 0))
index2, solution = BLS(list, index1, firstSolution)
optimal = neighborhood(index2, solution, firstSolution)
print("Best found of number of chicken sandwich,light chicken sandwich,vegetarian sandwich in order {} \nThe best benefit the company can get: {}".format(optimal[0:3],optimal[3]))
```

BASIC LOCAL SEARCH Output

```
Best found of number of chicken sandwich,light chicken sandwich,vegetarian sandwich in
    order [20, 18, 20]
The best benefit the company can get: 870
>
```

TABU SEARCH

```python
import random as rd
class TS():
    def __init__(self, tabu_tenure, max_coef, step):
        self.tabu_tenure = tabu_tenure
        self.max_coef=max_coef
        self.step=step
        self.Initial_solution = self.get_InitialSolution()
        self.tabu_str, self.Best_solution, self.Best_objvalue = self.TSearch()
    # Generate random initial solution
    def get_InitialSolution(self, show=False):
        X=rd.randint(1,self.max_coef)
        Y=rd.randint(1,self.max_coef)
        Z=rd.randint(1,self.max_coef)
        initial_solution=[X,Y,Z]
        if show == True:
            print("initial Random Solution: {}".format(initial_solution))
        return initial_solution
    # Objective function
    def Objfun(self, solution, show = False):
        '''
        f=10*X+15*Y+20*Z
        4X+2Y<=400
        2X+3Y+3Z<=300
        X+2Y+3Z<=200
        X+Y+4Z<= 900
        X,Y,Z>=0
        '''
        tabu=False
        X=solution[0]
        Y=solution[1]
        Z=solution[2]
        c1=4*X+2*Y
        if c1>400:
            tabu=True
        c2=2*X+3*Y+3*Z
        if c2>300:
            tabu=True
        c3=1*X+2*Y+3*Z
        if c3>200:
            tabu=True
        c4=1*X+1*Y+4*Z
        if c3>900:
            tabu=True
        c5=1*X+1*Y+1*Z
```

TABU SEARCH

```python
        c5=1*X+1*Y+1*Z
        if c5<0:
            tabu=True
        objfun_value=10*X+15*Y+20*Z
        if show == True:
            print("Best found of number of chicken sandwich,light chicken sandwich,vegetarian sandwich in order {} \nThe best benefit the company can get: {}\n".format(best_solution,best_objvalue))
        return objfun_value,tabu
    def GenertaeNeigbour(self, solution):
        '''
        Takes a  solution
        returns a new neighbor solution
        '''
        X=solution[0]
        Y=solution[1]
        Z=solution[2]
        solution = solution.copy()
        # Genertae neighbour
        # X,Y,Z are positives
        if (X-self.step)<0:
            a=1
        else:
            a=X-self.step
        solution[0]=rd.randint(a,X+self.step)
        if (Y-self.step)<0:
            a=1
        else:
            a=Y-self.step
        solution[1]=rd.randint(a,Y+self.step)
        if (Z-self.step)<0:
            a=1
        else:
            a=Z-self.step
        solution[2]=rd.randint(a,Z+self.step)
        return solution
    # Generate Tabu List
    def GenerateTabu(self, solution):
        dict = {}
        for _ in range(self.tabu_tenure):
            candidate_solution = self.GenertaeNeigbour(solution)
            candidate_objvalue,tabu = self.Objfun(candidate_solution)
            new_neibour=(candidate_solution[0],candidate_solution[1],candidate_solution[2])
            dict[new_neibour] = {'Value': candidate_objvalue, 'tabu': tabu }
```

TABU SEARCH

```python
        return dict
    def TSearch(self):
        '''The implementation Tabu search algorithm
        '''
        # Parameters:
        tenure =self.tabu_tenure
        best_solution = self.Initial_solution
        tabu_structure = self.GenerateTabu(best_solution)
        best_objvalue,_ = self.Objfun(best_solution)
        current_solution = self.Initial_solution
        current_objvalue = self.Objfun(current_solution)
        iter = 1
        Terminate = 0
        while Terminate < 100:
            # Searching the whole neighborhood of the current solution:
            tabu_structure=self.GenerateTabu(current_solution)
            # Admissible move
            while True:
                # select the solution with the highest ObjValue in the neighborhood (max)
                best_sol = max(tabu_structure, key =lambda x: tabu_structure[x]['Value'])
                SolValue = tabu_structure[best_sol]["Value"]
                tabu=tabu_structure[best_sol]["tabu"]
                # Not Tabu
                if not tabu :
                    # make it
                    current_solution = [best_sol[0],best_sol[1],best_sol[2]]
                    current_objvalue = SolValue
                    # Best Improving sol
                    if SolValue > best_objvalue:
                        best_solution = current_solution
                        best_objvalue = current_objvalue
                        Terminate = 0
                    else:
                        Terminate=Terminate+1
                        break
                # If tabu
                else:
                    tabu_structure[best_sol]["Value"] = -100
                    continue
        print("Best found of number of chicken sandwich,light chicken sandwich,vegetarian sandwich in order {} \nThe best benefit the company can get: {}\n".format(best_solution,best_objvalue))
        return tabu_structure, best_solution, best_objvalue
optimal = TS(tabu_tenure=200, max_coef=30, step=20)
```

TABU SEARCH OUTPUT

```
Best found of number of chicken sandwich,light chicken sandwich,vegetarian sandwich in
    order [96, 4, 32]
The best benefit the company can get: 1660

>
```

OUR DISCUSSION

Through what we saw in the results that appeared to us for both codes, we conclude that:
The second metaheuristic (Tabu search) is better because it gives us a greater objective function.

Thanks for LISTENING

ANY QUESTION?

BEDOOR AYDE 2005961
RIFAL KHALID 2006758