CS170 Spring 2017 — Project Writeup

Mike Lee, Jerry Park (SID: 26686553, 26517820)

Code

```
def solve(P, M, N, C, items, constraints):
  """
  Write your amazing algorithm here.

  Return: a list of strings, corresponding to item names.
  """
  maxer={}
  exp = 0.65


  itemProfit = 0
  itemWeight = 0
  itemCost = 0
  itemlist = []
  priorityQ = queue.PriorityQueue()
  random.shuffle(items)
  allconstraints = []
  restrictSet = set()

  # Counts the number of connections to other classes
  connections = np.ones(N)

  if constraints:
    for constraint in constraints:
      for c in constraint:
        connections[c] += len(constraint) - 1


  for i in items:
    # If this item restricts many other items if picked, decrease priority by division
    #exp = 0.4
    #
    # 0.65 for problem10, 0.75 for problem11, 0 for problem 12, 0.7 for problem13
    # 0 for problem14, 0.2 for problem15, 0.5 for problem16, 1 for problem17 and 18
    # 0 for problem19, 0.5 for problem20
```

```
    if i[4] - i[3] > 0:
      heuristic = (i[3]-i[4])/(connections[i[1]]**exp)
      priorityQ.put((heuristic,i))

  for i in range(N):
    #if(i%1000==0):
     # print(i)
    maxItem = priorityQ.get()[1]
    cond, temp = checkCons(maxItem[1], restrictSet, constraints)
    if cond:
      if itemWeight + maxItem[2] <= P and itemCost + maxItem[3] <= M:
        restrictSet = temp
        itemlist.append(maxItem[0])
        itemWeight += maxItem[2]
        itemCost += maxItem[3]
        itemProfit += maxItem[4]-maxItem[3]


  print ("Total Profit: ", itemProfit)


  return itemlist

def checkCons(lastItemAdded, restrictSet, constraints):
  if lastItemAdded in restrictSet:
    return (False, restrictSet)
  for constraint in constraints:
    if lastItemAdded in constraint:
      for item in constraint:
        if item != lastItemAdded:
          restrictSet.add(item)
  return (True, restrictSet)
      #check set to see if clash
```

Main Idea

This problem is similar to the Knapsack problem but in a more general version. The problem is also similar to an Independent Set problem because the constraints can be represented by a graph and the solution will only have classes that are an independent set of nodes of classes. To approach this problem with naive dynamic programming would yield exponential time and therefore to process 21 input files in a time-efficient manner, we implemented a greedy scheme which runs much quicker than exponential time. Depending on the input file, the runtime was from 3 seconds and occasionally up to a few minutes.

Our algorithm is as follows. Once we read in the input file and setting variables and lists accordingly, we created an array full of ones which represent each class and iterate through each constraint. For each class in this particular constraint, we added the length of the list to its value

in the array. This array basically represents how many classes this one particular class constrains and mathematically represents the degree of the class node if all constraints are represented as a graph.

We sorted the items with a priority queue with a heuristic (Resale Value - Cost)/(Degree in connectivity)^{exp}, where *exp* is the variable we change to find the most profitable set of items. This heuristic made the algorithm pick items that are more profitable and items that do not constrain us from picking other items before any other item. In the process of picking, we checked constraints and overweight and overspending along the way.

We ran values ranging from zero to two for the variable *exp* as different problem input had different optimal values for *exp*. The degree of connectivity improved the profit from a slight 10% to sometimes 150%.