Optimizing Dorm Assignment with Diversity Constraints

Yewon Doe, Ailin Yu, Sanjana Mahajan, Lantian Xu, Dinc Basar

December 16, 2019

1 Introduction

The goal of the dorm assignment project is to find an assignment to first year Carnegie Mellon students such that the total distance that students travel to their likely first class is minimized. This assignment must further abide by constraints that ensure diverse allocation within dorms. Minimizing average distance benefits students in having more time for daily activities and studying, while the diversity constraints allow students to have a socially productive first year, with an emphasis on distributing students to dorms in the most balanced way, according to their traits.

We start by modeling a simple minimization problem that does not take diversity constraints into account. Consequently, we try different algorithms that can expand the constraints, until we can include all the constraints that represent the real life scenario accurately.

2 Data Generation

Since we were not able to get data from the Housing Services for the 2019-2020 year, we had to come up with our own randomized method for generating accurate data. We got statistics for total number of students enrolled and the total number of student in each college and capacities for each of the freshmen dorms. There were a total of 1511 students from 6 different home colleges, CFA, DC, CIT, MCS, SCS and TPR each with 230, 298, 417, 222, 205 and 139 students. There were a total of 9 different dorms students could be assigned, BossHouse, Hamerschlag, Scobell, Donner, Morewood, Stever, Mudge, Rez and Shirley, each with capacities 70, 160, 84, 224, 414, 244, 288, 122, and 51. We set the gender equal to 55% male and 45 % female and randomly generated the data by setting the probability parameters. Moreover, we appended interest and preference columns which identify different constraints to the problem. The preferences were randomly generated by making permutation of the dorms. For

instance, a student will have his or her gender, major, interest, and 9 columns that shows the most preferred to the least preferred dorm choices.

Figure 1: Randomly Generated Data

gender	major	interests	first	second	third
0	1	4	5	8	6
0	2	8	1	8	5
1	2	3	8	5	1
1	3	6	1	3	2
1	3	6	6	0	8
1	3	2	8	6	2

3 Brute Force

At first, we attempted to use backtracking to assign students while keeping within the constraints. One by one, we attempted to assign students in a dorm that would keep the solution withing the constraints. When an nth student did not have any such dorms, we would backtrack to the last student and change their assignment, with the hopes that n^{th} student would now have a feasible assignment. However, we quickly realized that this might require the exhaustion of all options before terminating with an "infeasible" result if we set the constraints too tight. In other cases, it might still require the exploration of a vast majority of options in order to find a feasible solution. Since we have 9 dorms, and 1511 students, the runtime is $O(9^{1511})$, which is more than the number of atoms in the universe [1]. Therefore, after a day, we abandon this approach and move on with more efficient algorithmic considerations.

4 Python Linear Programming

We used a python package called `PuLP` to implement the linear programming algorithm which involves minimising the total distances subject to a set of linear inequality or equality constraints. We input a cost matrix which was defined in section 6.1 and the number of students in each major. In this algorithm, the students from the same major are treated identically, thus no individual traits can be implemented by this algorithm. We were able to take into account the capacity of each dorm, but when we added the constraint of filling at least 95% of each dorm our results is no longer integer so we had to round the results. The algorithm outputs a dorm-major matrix, Figure 2, that specifies how many students from each major should go to each dorm. The final total cost is 510.51 miles and the average cost per student is 0.34 miles

5 Greedy Algorithm

The possible assignment of dorm was impossible to solve since the space of all solution was about 9^{1511}. Therefore, for greedy algorithm we only took the

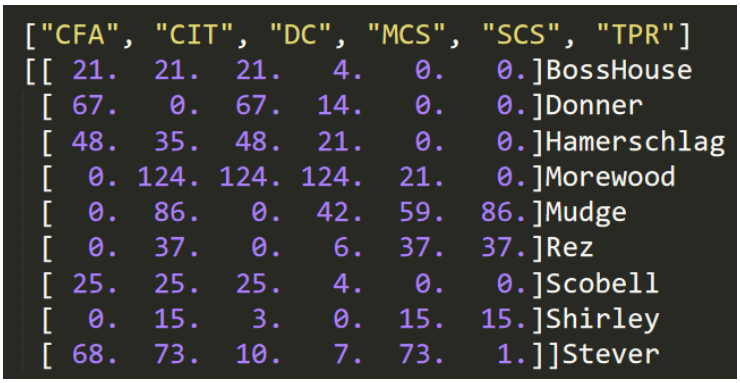

Figure 2: Linear Programming Algorithm Output Matrix

preference columns as objective function and capacity of dorms as the constraint and found one possible solution to the problem. The cost for the preference was their preference number divided by 20 so if a student gets their first preference the cost for the objective function will be added by $\frac{1}{20}$. The algorithm first started giving students their first preference until the capacity constraint was not violated and then started assigning the second preference when the dorms were full. We achieved around 115.3 cost which indicates that we were able to assign students with average 0.1 cost per student, which is on average they got their second preference.

6 Hungarian Algorithm

The Hungarian algorithm offers a one to one match that minimizes the cost while taking the capacity of each dorm into account. We used a python package called `munkres`, which requires us to input a cost matrix, to implement the algorithm.

6.1 Cost Matrix

The cost matrix is an n by n matrix where n represents the number of students as well as the number of beds available (in this case the total capacity of dorms). However, in our original dataset, the capacity is higher than the number of students. To solve this problem, we reduced the capacity of Boss House is reduced from 70 to 35 and the capacity of Morewood Gardens is decreased by 11 since these are the two dorms that are for not only freshmen but also upper-class students to live in. Now the cost matrix is 1511 by 1511, and we filled in the entries such that each entry is the sum of distance and preference score. In this way, we make sure that we are minimizing distance and satisfying student preferences at the same time. A part of the cost matrix looks as follows:

Figure 3: Hungarian Algorithm cost matrix

Student	Boss_House	Boss_House	Boss_House	Boss_House
1	0.35	0.35	0.35	0.35
2	0.5	0.5	0.5	0.5
3	0.25	0.25	0.25	0.25
4	0.5	0.5	0.5	0.5
5	0.45	0.45	0.45	0.45
6	0.4	0.4	0.4	0.4
7	0.45	0.45	0.45	0.45
8	0.4	0.4	0.4	0.4
9	0.45	0.45	0.45	0.45
10	0.6	0.6	0.6	0.6
11	0.65	0.65	0.65	0.65

6.2 Hungarian Output

The algorithm outputs which bed each student should be assigned to, and we convert the output the number of students in each college that are assigned to each dorm by looking at each student's major and where each bed belongs to. The final output is presented below.

Figure 4: Hungarian Algorithm Output

	CFA	DC	CIT	MCS	SCS	TPR
Boss_House	21	12	0	2	0	0
Hamerschlag	64	54	17	25	0	0
Scobell	32	40	5	7	0	0
Donner	60	75	58	30	1	0
Morewood	24	43	91	58	63	24
Stever	20	29	70	42	57	26
Mudge	9	41	81	52	63	42
Rez	0	4	56	6	13	43
Shirley	0	0	39	0	8	4

According to the solution, No CFA student lives in Residence on Fifth or Shirley Apartments, which is reasonable since those are the two dorms that locate farthest from the Purnell Center for the Arts. However, we do see 4 Tepper students are assigned to the faraway dorm Shirley, which demonstrates

how preference also plays a role in the decision making process.

7 LP Relaxation Using Simplex

In order to capture the full scope of our minimization problem and constraints, we decide to model it as a linear program and use the Simplex Algorithm to find non-integer solutions. Since we require solutions to be integers (we cannot divide students in pieces), we employ a backtracking algorithm on the results of the LP relaxation to find another solution within the feasible region. We then analyze the optimality of this solution, which ends up being significantly close to the theoretical optimum.

7.1 Variables

We first define our variables to be x_1 ... x_{mn}, where m is the number of dorms and n is the number of students. Each student therefore gets m many variables, x_i ... x_{i+m-1}, each of which represents the probability that student i is assigned to dorm j.

7.2 Objective Function

We then define the cost function to be the expected distance of a student's assignment to their college building, which is where their first class likely will be. This expected distance is given by

$$\sum_{j \in \text{ dorms}} \Pr[i \text{ is assigned to dorm } j] * \text{ distance}(i\text{'s major, dorm } j) \quad (1)$$

Therefore, we create the vector $\vec{d}$, of length $1 \times mn$, which represents the distance. We then take the dot product of d and x, which our objective is to minimize. This can be represented as the summation

$$\sum_{k=1}^{mn} d_k x_k \quad (2)$$

7.3 Constraints

The first two constraints will ensure that we can make a priority queue from the probabilities and assign the student to the highest feasible probability. The next two constraints are our initial diversity constraints.

1. Each student can get at most 1 assignment in probabilities.

$$\sum_{k=i}^{i+m-1} x_k \leq 1 \text{ for all } i = 1, m+1, \ldots, mn-m+1 \tag{3}$$

2. Each student can get at least 1 assignment in probabilities.

$$\sum_{k=i}^{i+m-1} x_k \geq 1 \text{ for all } i = 1, m+1, \ldots, mn-m+1 \tag{4}$$

3. Each dorm can get at most 55% of any gender. Here G_i is the gender of student i and C_j is the capacity of dorm j.

$$\sum_{i=1}^{n} I(G_i = G) \cdot x_{m(i-1)+j} \leq 0.55 C_j \text{ for all dorms } j \in [m] \text{ and genders } G \tag{5}$$

4. Each dorm can get at most 30% of any major. Here M_i is the major of student i and C_j is the capacity of dorm j.

$$\sum_{i=1}^{n} I(M_i = M) \cdot x_{m(i-1)+j} \leq 0.3 C_j \text{ for all dorms } j \in [m] \text{ and majors } M \tag{6}$$

In this setting, the Simplex Algorithm terminates in less than 30 minutes, as it pivots tens of thousands of times. However, this is still a significant improvement over any of the other algorithms we tried, as it captures the entirety of the problem, finds an almost optimal solution, and is able to terminate before the universe comes to an end.

```
For 1511 students, optimum is 577.305
Average is 0.3821 per student
  Dorm Name Num Lim     Percent               CFA  DC CIT MCS SCS TPR
       Boss  70  70 (100.00% full) --> majors: [10, 18, 17, 10, 10,  5]
Hamerschlag 160 160 (100.00% full) --> majors: [39, 30, 19, 24, 27, 21]
    Scobell  85  84 (101.19% full) --> majors: [15, 12, 26, 10, 14,  8]
     Donner 224 224 (100.00% full) --> majors: [28, 42, 67, 36, 36, 15]
   Morewood 314 314 (100.00% full) --> majors: [43, 79, 76, 44, 42, 30]
     Stever 244 244 (100.00% full) --> majors: [42, 39, 74, 38, 26, 25]
      Mudge 289 288 (100.35% full) --> majors: [32, 60, 87, 49, 33, 28]
        Rez 107 122 ( 87.70% full) --> majors: [20, 16, 36, 11, 17,  7]
    Shirley  18  51 ( 35.29% full) --> majors: [ 1,  2, 15,  0,  0,  0]
```

8 Extensions to LP Relaxation

8.1 Filling At Least 95% of Each Dorm

With the initial setup of the linear problem, we realized that the farther dorms get the least assignment possible, as can be seen in the figure above. For instance, Shirley has only 18 students assigned out of a capacity of 51, while 7 other dorms are full to the brink. This is not ideal for the Housing Services, as they would like to distribute the load evenly so that none of the facilities are strained to the max, if not need be. Therefore, we add an additional constraint that ensures at least 95% of each dorm is filled, which results in a much more evenly spread distribution of students.

$$\sum_{i=1}^{n} x_{m(i-1)+j} \geq 0.95 C_j \text{ for all dorms } j \in [m] \tag{7}$$

```
For 1511 students, optimum is 592.41
Average is 0.3921 per student
  Dorm Name Num Lim     Percent               CFA  DC CIT MCS SCS TPR
       Boss  70  70 (100.00% full) --> majors: [ 5, 11, 21, 21,  6,  6]
Hamerschlag 160 160 (100.00% full) --> majors: [34, 29, 42, 27, 19,  9]
    Scobell  83  84 ( 98.81% full) --> majors: [25,  9, 10, 25,  7,  7]
     Donner 224 224 (100.00% full) --> majors: [39, 34, 67, 30, 27, 27]
   Morewood 303 314 ( 96.50% full) --> majors: [45, 67, 94, 35, 39, 23]
     Stever 232 244 ( 95.08% full) --> majors: [17, 52, 73, 29, 43, 18]
      Mudge 274 288 ( 95.14% full) --> majors: [44, 65, 58, 34, 41, 32]
        Rez 116 122 ( 95.08% full) --> majors: [18, 26, 37,  5, 18, 12]
    Shirley  49  51 ( 96.08% full) --> majors: [ 3,  5, 15, 16,  5,  5]
```

It can be seen that only 3 of the dorms are full now, and we do not even need to run the integer solution conversion as none of the dorms have more students than they have capacity for.

8.2 Student Preferences for Dorms

Each incoming Carnegie Mellon first-year submits a form indicating their preferred dormitory assignments. While it is not possible to accommodate all of these preferences, the Housing Services has significant interest in ensuring that students get the highest pick available to them. Therefore, we capture this within our objective function, which is now to minimize the distance for each student, as well as the cost incurred by assigning them their ith preference. We set the cost to be 0.05 per preference, to closely align with the cost of distances in miles, so that if a student gets their first preference, the cost would be 0, and their 5th preference would be 0.25, and so on. Let $\vec{p}$, of length $1 \times mn$, represent these preference costs. Our new objective function is

$$\sum_{k=1}^{mn} (d_k + p_k) x_k \tag{8}$$

```
For 1511 students, optimum is 613.795
Average is 0.4062 per student
  Dorm Name Num Lim     Percent               CFA  DC CIT MCS SCS TPR
       Boss  71  70 (101.43% full) --> majors: [17, 13, 14, 11,  9,  7]
Hamerschlag 160 160 (100.00% full) --> majors: [32, 29, 35, 21, 24, 19]
    Scobell  84  84 (100.00% full) --> majors: [13, 17, 25,  7, 11, 11]
     Donner 223 224 ( 99.55% full) --> majors: [27, 52, 67, 28, 34, 15]
   Morewood 300 314 ( 95.54% full) --> majors: [39, 64, 84, 49, 39, 25]
     Stever 235 244 ( 96.31% full) --> majors: [37, 39, 73, 36, 27, 23]
      Mudge 273 288 ( 94.79% full) --> majors: [45, 52, 74, 44, 37, 21]
        Rez 116 122 ( 95.08% full) --> majors: [14, 27, 30, 16, 15, 14]
    Shirley  49  51 ( 96.08% full) --> majors: [ 6,  5, 15, 10,  9,  4]
```

8.3 Interest Categories

First year is important in terms of incoming students making friends that will last for the duration of their undergraduate studies. Therefore, we introduce an extra feature to our algorithm: interest categories. We uniform randomly pick an integer [0-9] for each student, indicating the types of their interests. Examples to this can be arts, sports, outdoors, music and more. Due to the uniform random distribution, each category is expected to have 10% of the student body, so we add an extra constraint that each dorm must have at least 5% of each interest category. This means that regardless of the dorm that a students ends up in, there will be at least a couple of other students in the same interest category that they can potentially be friends with and get involved in activities together. Let the quantity H_i represent the interests (or hobbies) of student i. As before, C_j is the capacity of dorm j. The constraint capturing this idea is

$$\sum_{i=1}^{n} I(H_i = H) \cdot x_{m(i-1)+j} \geq 0.05 C_j \text{ for all dorms } j \in [m] \text{ and interests } H \quad (9)$$

We tried running the simplex algorithm by increasing the constraint to at least 7% of each category; however, the algorithm did not terminate in more than 12 hours. Therefore, we take into account the former constraint, which has a negligible difference to the latter constraint in terms of number of students.

9 LP Relaxation to Integer Solution

After the Simplex Algorithm outputs a solution, we use backtracking to obtain an integral solution. In order to complete in an acceptable run time, we do not exhaust the entire search space, but we still find an almost solution that also meets the constraints we introduced. We usually have 2-3 students that breach the constraints, and we start by finding the maximum major distribution within the dorms that are over-filled, and the minimum major distributions within the dorms that are least filled. We then smartly place students to the

least filled dorms, in order to stay within the feasible region. While this results in a non-optimal solution, the cost increase of 2-3 students is usually less than 0.5, in comparison to the approximately 600 total cost. Therefore, we know that our solution is within 1.001-opt, making it negligibly different. This is a valid trade-off for run-time sake.

Before:

```
For 1511 students, optimum is 613.795
Average is 0.4062 per student
  Dorm Name Num Lim     Percent                CFA  DC CIT MCS SCS TPR
       Boss  71  70 (101.43% full) --> majors: [17, 13, 14, 11,  9,  7]
Hamerschlag 160 160 (100.00% full) --> majors: [32, 29, 35, 21, 24, 19]
    Scobell  84  84 (100.00% full) --> majors: [13, 17, 25,  7, 11, 11]
     Donner 223 224 ( 99.55% full) --> majors: [27, 52, 67, 28, 34, 15]
   Morewood 300 314 ( 95.54% full) --> majors: [39, 64, 84, 49, 39, 25]
     Stever 235 244 ( 96.31% full) --> majors: [37, 39, 73, 36, 27, 23]
      Mudge 273 288 ( 94.79% full) --> majors: [45, 52, 74, 44, 37, 21]
        Rez 116 122 ( 95.08% full) --> majors: [14, 27, 30, 16, 15, 14]
    Shirley  49  51 ( 96.08% full) --> majors: [ 6,  5, 15, 10,  9,  4]
```

After:

```
For 1511 students, optimum is 614.09
Average is 0.4064 per student
  Dorm Name Num Lim     Percent                CFA  DC CIT MCS SCS TPR
       Boss  70  70 (100.00% full) --> majors: [16, 13, 14, 11,  9,  7]
Hamerschlag 160 160 (100.00% full) --> majors: [32, 29, 35, 21, 24, 19]
    Scobell  84  84 (100.00% full) --> majors: [13, 17, 25,  7, 11, 11]
     Donner 223 224 ( 99.55% full) --> majors: [27, 52, 67, 28, 34, 15]
   Morewood 300 314 ( 95.54% full) --> majors: [39, 64, 84, 49, 39, 25]
     Stever 235 244 ( 96.31% full) --> majors: [37, 39, 73, 36, 27, 23]
      Mudge 274 288 ( 95.14% full) --> majors: [46, 52, 74, 44, 37, 21]
        Rez 116 122 ( 95.08% full) --> majors: [14, 27, 30, 16, 15, 14]
    Shirley  49  51 ( 96.08% full) --> majors: [ 6,  5, 15, 10,  9,  4]
```

It can be seen that the latter is within 0.002 of the theoretical optimum, which is the solution to the LP Relaxation problem.

10 Conclusions and Limitations

After exploring many methods for solving our problem, the Simplex Algorithm turned out to be the most powerful one. It allowed us to not only incorporate all of the base constraints, but also consider extensions to the main problem. The other methods we considered involved using in-built Python packages that were simply not flexible enough to capture the scope of the problem. In regards to further development, we would consider making our implementation of the Simplex Algorithm slightly more efficient, so that we can include even

more constraints if necessary. Finally, if we can get in contact with the right people in CMU Housing Services and learn about the data formats they use, we can transform our algorithm to incorporate that data format, and provide the Housing Services with a tool that they can use every year reliably.

References

[1] Vishal Thakur. How many atoms are in the universe?

11 Code

11.1 Data Generation

```
import numpy as np
import pandas as pd
total = []

np.random.seed(393)

majors = np.array([225,275,411,232,211,157])
num_students = sum(majors)
num_majors = len(majors)
num_interests = 10
num_dorms = 9
f = lambda x: x / num_students
major_probabilities = f(majors)

df = pd.DataFrame([], columns=['gender', 'major', 'interests'])

gender = np.random.choice(2, num_students, p=[0.55, 0.45])
df["gender"] = gender

nums = [0] * num_majors
for i in range(num_majors):
  nums[i] = majors[i] / num_students

major = np.random.choice(num_majors, num_students, p=major_probabilities)
df["major"] = major

interests = np.random.choice(num_interests, num_students)
df["interests"] = interests

all_preference = []
for i in range(num_students):
    all_preference.append(np.random.permutation(num_dorms))
preference = pd.DataFrame(all_preference)
preference.columns = ["first","second","third","fourth","fifth","sixth","seventh", "eighth",

df = pd.concat([df,preference], axis=1)

df.to_csv("students.csv")
```

11.2 Brute Force

11.3 Python Linear Programming

11.4 Greedy Algorithm

```
def check_capacity(df):
    a = df.groupby(["select"]).groups
    capacity = [70,160,224,414,288,122,84,51,244]
    for elem in a:
        if len(a[elem]) > capacity[int(elem)]:
            return False
    return True

def main(df,weight):
    n = df.shape[0]
    for i in range(n):
        df.loc[i,"select"] = df.loc[i,"first"]
        df.loc[i,"preference"] = 1
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"second"]
            df.loc[i,"preference"] = 2
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"third"]
            df.loc[i,"preference"] = 3
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"fourth"]
            df.loc[i,"preference"] = 4
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"fifth"]
            df.loc[i,"preference"] = 5
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"sixth"]
            df.loc[i,"preference"] = 6
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"seventh"]
            df.loc[i,"preference"] = 7
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"eighth"]
            df.loc[i,"preference"] = 8
        if not check_capacity(df.loc[:(i+1),]):
            df.loc[i,"select"] = df.loc[i,"ninth"]
            df.loc[i,"preference"] = 9
    if check_capacity(df):
        num = sum(df["preference"]) / 20
    return df,num
```

11.5 Hungarian Algorithm

```
import pandas as pd
import numpy as np

cost = pd.read_csv("cost_matrix.csv")
my_cost = cost.iloc[:,2:1513]
matrix = np.array(my_cost).tolist()
from munkres import Munkres, print_matrix

m = Munkres()
indexes = m.compute(matrix)
total = 0
count = 0
solution = []
for row, column in indexes:
    value = matrix[row][column]
    total += value
    count += 1
    solution.append((row, column))
print(f'total cost: {total}')

student_dorm = []
dorms = list(my_cost.columns)
for (student, bed) in solution:
    dorm = dorms[bed]
    my_dorm = dorm.split(".", maxsplit = 1)[0]
    student_dorm.append((student, my_dorm))

majors = cost.iloc[:,1]
colleges = ["CFA", "DC", "CIT", "MCS", "SCS", "TPR"]
major_dorm = []
college_dorm = []
for (student, dorm) in student_dorm:
    major = majors[student]
    major_dorm.append((major, dorm))
    my_college = colleges[major]
    college_dorm.append((my_college, dorm))

unique_dorm = ["Boss_House", "Hamerschlag", "Scobell", "Donner", "Morewood", "Stever", "Mudg
output = []
for row in range(len(unique_dorm)):
```

```
    output += [[0]*len(colleges)]

output
```

11.6 Linear Programming with PuLP

The following code is adopted from The Beer Distribution Problem for the PuLP Modeller, original authors: Antony Phillips, Dr Stuart Mitchell 2007.

```
from pulp import *
import numpy as np
import math

def my_round(i):
  f = math.floor(i)
  if i - f < 0.5:
    return f
  else:
    return f+1

Dorms = ["BossHouse", "Hamerschlag","Scobell",
 "Donner", "Morewood", "Stever",
 "Mudge","Rez","Shirley"]


Dorm_Total = {"BossHouse":70, "Hamerschlag":160, "Scobell":84,
"Donner":224, "Morewood":414, "Stever":244, "Mudge":288,
 "Rez":122, "Shirley":51}

Colleges = ["CFA", "DC", "CIT", "MCS", "SCS","TPR"]

College_Total = {"CFA":230,"DC":298,"CIT":417,"MCS":222,"SCS":205,"TPR":139}

Students_Total = sum(College_Total.values())

College_Proportions = {i: t / Students_Total for i, t in College_Total.items()}

Dorm_Capacity = {}
for i,t in Dorm_Total.items():
    tmp = []
    for c in Colleges:
        tmp.append(math.ceil(College_Proportions[c]*t))
    Dorm_Capacity[i] = tmp

costs = [#majors
```

```
            [0.2, 0.3,0.4,0.3,0.4,0.4], #Boss Dorms
            [0.2,0.3,0.4,0.3,0.4,0.4], #Hamerschleg
           [0.2,0.3,0.4,0.3,0.4,0.4],#Scobell
            [0.15,0.25,0.35,0.25,0.35,0.35], #Donner
           [0.35,0.4,0.4,0.3,0.2,0.06],#Morewood
            [0.4,0.45,0.45,0.35,0.25,0.1],#Setever
           [0.5,0.5,0.5,0.4,0.3,0.1],#Mudge
           [0.7,0.7,0.6,0.6,0.5,0.2],#Rez
            [0.9, 0.9, 0.7, 0.8,0.6,0.4],#Shirley
            ]

costs = makeDict([Dorms, Colleges],costs,0)

prob = LpProblem("Dorm_Assignment_Problem",LpMinimize)

Routes = [(w,b) for w in Dorms for b in Colleges]

vars = LpVariable.dicts("Route", (Dorms, Colleges),0,None,LpInteger)


prob += lpSum([vars[w][b]*costs[w][b] for (w,b) in Routes]), "Sum_of_Transportin


for w in Dorms:
    prob += lpSum([vars[w][b] for b in Colleges]) <= Dorm_Total[w], "Sum_of_Proc
for w in Dorms:
    for (i,b) in enumerate(Colleges):
        prob += vars[w][b] <= 0.3*Dorm_Total[w], "Major_%s_Dorm_%s"%(b,w)
    prob += lpSum([vars[w][c] for c in Colleges]) >= 0.95*Dorm_Total[w], "Dorm_%

College_Total_constraint = {}
for c in Colleges:
    constraint = lpSum([vars[d][c] for d in Dorms]) == College_Total[c]
    prob += constraint, "Sum_of_Products_into_College_%s"%c
    College_Total_constraint[c] = constraint
prob.writeLP("Dorm_Assignment_Problem.lp")
final = np.zeros(54)
final_dict = {}
for d in Dorms:
    final_dict[d] = {}
for college_total in range(0,College_Total["CFA"],1):
    College_Total_constraint["CFA"].constant = - college_total
    prob.solve()
    for (i, v) in enumerate(prob.variables()):
        final[i] = my_round(v.varValue)
        _,dorm, major = v.name.split("_")
```

```
        final_dict[dorm][major] = my_round(v.varValue)
print("Total Cost of Transportation =", value(prob.objective))
print("Average Cost per Student =", value(prob.objective)/Students_Total)

Colleges = ["CFA", "CIT", "DC", "MCS", "SCS", "TPR"]
Dorms = ["BossHouse","Donner", "Hamerschlag","Morewood",
 "Mudge", "Rez", "Scobell",
 "Shirley","Stever"]

final = final.reshape(9,6)
print(final)
major_sums = {}
for c in Colleges:
    major_sums[c] = 0
#print(final_dict)
for v in final_dict:
    total = sum(final_dict[v].values())
    t = (total <= Dorm_Total[v])
    print("Dorm %s Total Constraint Satisfied %r"%(v, t))
    for m in final_dict[v]:
        major_sums[m] += final_dict[v][m]
for i in major_sums:
    t = (College_Total[i] >= major_sums[i])
    print("Major %s Total Constraint Satisfied %r"%(i, t))

'''

["CFA", "CIT", "DC", "MCS", "SCS", "TPR"]
[[ 21.  21.  21.   4.   0.   0.]BossHouse
 [ 67.   0.  67.  14.   0.   0.]Donner
 [ 48.  35.  48.  21.   0.   0.]Hamerschlag
 [  0. 124. 124. 124.  21.   0.]Morewood
 [  0.  86.   0.  42.  59.  86.]Mudge
 [  0.  37.   0.   6.  37.  37.]Rez
 [ 25.  25.  25.   4.   0.   0.]Scobell
 [  0.  15.   3.   0.  15.  15.]Shirley
 [ 68.  73.  10.   7.  73.   1.]]Stever

'''
```

11.7 LP Relaxation - Simplex

```
#include <iostream>
#include <fstream>
```

```
#include <vector>
#include <sstream>
#include <string>
#include <simplex.h>

using namespace std;

vector<vector<double>> read_record()
{
  ifstream infile("students.csv");

  vector<vector<double>> students;
  vector<double> row;

  string line;

  while (infile >> line)
  {
    row.clear();

    // used for breaking words
    stringstream s(line);

    string entry_str;

    int i = 0;
    while (getline(s, entry_str, ','))
    {
      if (i == 0)
      {
        i++;
        continue;
      }

      row.push_back(stof(entry_str));

      i++;

      entry_str.clear();
    }

    students.push_back(row);
  }

  return students;
}
```

```
vector<vector<double>> make_A(vector<vector<double>> students, int num_students,
    vector<int> capacities, int num_dorms, int num_majors, int num_interests)
{
  vector<vector<double>> A;

  int cols = num_students * num_dorms;

  vector<double> row;

  int lower_bd, upper_bd;

  for (int i = 0; i < num_students; i++)  // all students get an assignment
  {
    row.clear();

    lower_bd = i * num_dorms;
    upper_bd = (i + 1) * num_dorms;

    for (int j = 0; j < cols; j++)
    {
      if (j >= lower_bd && j < upper_bd)
      {
        row.push_back(1.0);
      }
      else
      {
        row.push_back(0.0);
      }
    }
    A.push_back(row);

    row.clear();
    for (int j = 0; j < cols; j++)
    {
      if (j >= lower_bd && j < upper_bd)
      {
        row.push_back(-1.0);
      }
      else
      {
        row.push_back(0.0);
      }
    }
    A.push_back(row);
  }
```

```
for (int i = 0; i < num_dorms; i++)    // all dorms get less than capacity
{
  row.clear();
  for (int j = 0; j < cols; j++)
  {
    if (j % num_dorms == i)
    {
      row.push_back(1.0);
    }
    else
    {
      row.push_back(0.0);
    }
  }
  A.push_back(row);
}

for (int i = 0; i < num_dorms; i++)    // all dorms get more than 90% capacity
{
  row.clear();
  for (int j = 0; j < cols; j++)
  {
    if (j % num_dorms == i)
    {
      row.push_back(-1.0);
    }
    else
    {
      row.push_back(0.0);
    }
  }
  A.push_back(row);
}

int gender;
for (int i = 0 ; i < num_dorms; i++)  // gender distribution per dorm
{
  row.clear();
  for (int j = 0; j < cols; j++)
  {
    gender = students[j / num_dorms][0];  // j/num_dorms'th student
    if (j % num_dorms == i)
    {
      if (gender == 0.0)
      {
```

```
        row.push_back(1.0);
      }
      else
      {
        row.push_back(0.0);
      }
    }
    else
    {
      row.push_back(0.0);
    }
  }

  A.push_back(row);

  row.clear();
  for (int j = 0; j < cols; j++)
  {
    gender = students[j / num_dorms][0];     // j/num_dorms'th student
    if (j % num_dorms == i)
    {
      if (gender == 1.0)
      {
        row.push_back(1.0);
      }
      else
      {
        row.push_back(0.0);
      }
    }
    else
    {
      row.push_back(0.0);
    }
  }
  A.push_back(row);
}

int major;
for (int i = 0 ; i < num_dorms; i++)    // major distribution per dorm
{
  for (int k = 0; k < num_majors; k++)
  {
    row.clear();
    for (int j = 0; j < cols; j++)
    {
```

```
        major = students[j / num_dorms][1];    // j/num_dorms'th student
        if (j % num_dorms == i && major == (double) k)
        {
          row.push_back(1.0);
        }
        else
        {
          row.push_back(0.0);
        }
      }
      A.push_back(row);
    }
  }

  int interest;
  for (int i = 0 ; i < num_dorms; i++)        // interest distribution per dorm
  {
    for (int k = 0; k < num_interests; k++)
    {
      row.clear();
      for (int j = 0; j < cols; j++)
      {
        interest = students[j / num_interests][2];     // j/num_dorms'th student
        if (j % num_dorms == i && interest == (double) k)
        {
          row.push_back(-1.0);
        }
        else
        {
          row.push_back(0.0);
        }
      }
      A.push_back(row);
    }
  }

  return A;
}

vector<double> make_B(vector<vector<double>> students, int num_students,
    vector<int> capacities, int num_dorms, int num_majors, int num_interests)
{
  vector<double> B;

  for (int i = 0; i < num_students; i++)
  {
```

```
  B.push_back(1.0);
  B.push_back(-1.0);
}

for (int i = 0; i < num_dorms; i++)
{
  B.push_back((double) capacities[i]);
}

double ratio, alpha;

alpha = -0.95;
for (int i = 0; i < num_dorms; i++)
{
  ratio = (double) capacities[i] * alpha;
  B.push_back(ratio);
}

alpha = 0.55;
for (int i = 0; i < num_dorms; i++)
{
  ratio = (double) capacities[i] * alpha;
  B.push_back(ratio);
  B.push_back(ratio);
}

alpha = 0.3;
for (int i = 0; i < num_dorms; i++)
{
  for (int k = 0; k < num_majors; k++)
  {
    ratio = capacities[i] * alpha;
    B.push_back(ratio);
  }
}

alpha = -0.05;
for (int i = 0; i < num_dorms; i++)
{
  for (int k = 0; k < num_interests; k++)
  {
    ratio = capacities[i] * alpha;
    B.push_back(ratio);
  }
}
```

```
  return B;
}

vector<double> make_C(vector<vector<double>> distances,
    vector<vector<double>> students,
    int num_students, int num_dorms, int mode)
{
  vector<double> C;
  int major;
  double coefficient;

  for (int i = 0; i < num_students; i++)
  {
    major = students[i][0];

    for (int dorm_id = 0; dorm_id < num_dorms; dorm_id++)
    {
      if (mode == 0)
      {
        coefficient = distances[dorm_id][major];
      }
      else if (mode == 1)
      {
        coefficient = distances[dorm_id][major] + students[i][dorm_id + 3];
        // distance + preference coeff
      }
      else if (mode == 2)
      {
        coefficient = students[i][dorm_id + 3];
      }
      C.push_back(-1.0 * coefficient);      // min cx = max -cx
    }
  }

  return C;
}

int main(int argc, char** argv)
{
  auto start = std::chrono::system_clock::now();

  vector<vector<double>> students = read_record();

  vector<int> capacities = {
                              70,     // Boss House
                              160,    // Hamerschlag
```

```
                            84,     // Scobell
                            224,    // Donner
                            314,    // Morewood
                            244,    // Stever
                            288,    // Mudge
                            122,    // Rez
                            51,     // Shirley
                          };

vector<vector<double>> distances = {
             //    CFA   DC    CIT   MCS   SCS   TPR
                  {0.2,  0.3,  0.4,  0.3,  0.4,  0.4  },      // Boss House
                  {0.2,  0.3,  0.4,  0.3,  0.4,  0.4  },      // Hamerschlag
                  {0.2,  0.3,  0.4,  0.3,  0.4,  0.4  },      // Scobell
                  {0.15, 0.25, 0.35, 0.25, 0.35, 0.35 },      // Donner
                  {0.35, 0.4,  0.4,  0.3,  0.2,  0.06 },      // Morewood
                  {0.4,  0.45, 0.45, 0.35, 0.25, 0.1  },      // Stever
                  {0.5,  0.5,  0.5,  0.4,  0.3,  0.1  },      // Mudge
                  {0.7,  0.7,  0.6,  0.6,  0.5,  0.2  },      // Rez
                  {0.9,  0.9,  0.7,  0.8,  0.6,  0.4  }       // Shirley
                  }; // distances in miles


int num_majors = distances[0].size();
int num_interests = 10;
int num_dorms = capacities.size();
int num_students, mode;
num_students = students.size();
mode = std::stoi(argv[1]);

printf("for num_students: %d\n", num_students);

vector<vector<double> > A = make_A(students, num_students, capacities,
                                   num_dorms, num_majors, num_interests);

vector<double> B = make_B(students, num_students, capacities,
                          num_dorms, num_majors, num_interests);

vector<double> C = make_C(distances, students, num_students, num_dorms, mode);

printf("sizes --> A: %lu x %lu, B: %lu, C: %lu\n",
  A.size(), A[0].size(), B.size(), C.size());

Simplex lp(A.size(), A[0].size(), A, B, C);

// std::cout << pivots << std::endl;
```

```
  if (lp.lp_type == lp.UNBOUNDED) {
      std::cout << "unbounded" << std::endl;
  } else if (lp.lp_type == lp.INFEASIBLE) {
      std::cout << "infeasible" << std::endl;
  } else if (lp.lp_type == lp.FEASIBLE) {
      std::cout << "The optimum is " << (-1 * lp.z) << std::endl;
      for (int i = 0; i<A[0].size(); i++) {
          std::cout << lp.soln[i] << std::endl;
      }
  } else {
      std::cout << "Should not have happened" << std::endl;
  }

  auto end = std::chrono::system_clock::now();
  std::chrono::duration<double> elapsed_seconds = end-start;
  std::cout << "elapsed time: " << elapsed_seconds.count() << "s\n";

  return 0;
}
```

11.8 LP Relaxation to Integer Solution

```
import numpy as np
import sys

if __name__ == "__main__":
    path = sys.argv[1]

capacities = [
              70,
              160,
              84,
              224,
              314,
              244,
              288,
              122,
              51
            ]

distances = [
```

```
            [0.2,  0.3,  0.4,  0.3,  0.4,  0.4  ],      # Boss Dorms
            [0.2,  0.3,  0.4,  0.3,  0.4,  0.4  ],      # Hamerschlag
            [0.2,  0.3,  0.4,  0.3,  0.4,  0.4  ],      # Scobell
            [0.15, 0.25, 0.35, 0.25, 0.35, 0.35 ],      # Donner
            [0.35, 0.4,  0.4,  0.3,  0.2,  0.06 ],      # Morewood
            [0.4,  0.45, 0.45, 0.35, 0.25, 0.1  ],      # Stever
            [0.5,  0.5,  0.5,  0.4,  0.3,  0.1  ],      # Mudge
            [0.7,  0.7,  0.6,  0.6,  0.5,  0.2  ],      # Rez
            [0.9,  0.9,  0.7,  0.8,  0.6,  0.4  ]       # Shirley
            ];

fd = open(path, mode = "r")
lines = fd.readlines()

num_students = int(lines[0].split(":")[-1][1:])

def find_opt():
    opt_str = "The optimum is"
    for i in range(len(lines)):
        line = lines[i]
        if (line[:len(opt_str)] == opt_str):
            opt_index = i
    optimum = float(lines[opt_index].split(" ")[-1])


    return optimum

optimum = find_opt()

num_dorms = 9
num_majors = 6

majors = []
fd2 = open("students.csv", mode = "r")
for row in fd2.readlines():
    majors.append(int(row.split(",")[2]))

picks = []

lines = lines[3:]

dorms = [[0] * num_majors for i in range(num_dorms)]

for i in range(num_students):
    student_range = lines[i * num_dorms : (i+1) * num_dorms]
```

```
    pick = student_range.index(max(student_range))

    dorms[pick][majors[i]] += 1

    # optimum += distances[pick][majors[i]]

    # print("Student", i, "dorm:", pick)

    picks.append(pick)



for i in range(len(dorms)):
    num_assigned = sum(dorms[i])

    while (num_assigned > capacities[i]):

        least_index = -1;
        least_ratio = 2;

        for j in range(len(dorms)):
            if (i == j):
                continue

            ratio = sum(dorms[j]) / capacities[j]

            if (ratio < least_ratio):
                least_index = j
                least_ratio = ratio

        max_index = dorms[i].index(max(dorms[i]))
        # min_index = dorms[least_index].index(min(dorms[least_index]))

        dorms[i][max_index] -= 1
        dorms[least_index][max_index] += 1
        num_assigned -= 1

        optimum -= distances[i][max_index]
        optimum += distances[least_index][max_index]

        # print(distances[i][max_index], distances[j][min_index])


names = ["Boss", "Hamerschlag", "Scobell", "Donner", "Morewood",
            "Stever", "Mudge", "Rez", "Shirley"]
```

```
print("For", num_students, "students, optimum is %.2f" % optimum)
print("Average is %.4f per student" % (optimum / num_students))

print("%11s %03s %03s %11s                %3s %3s %3s %3s %3s %3s" %
            ("Dorm Name", "Num", "Lim", "Percent", "CFA", "DC", "CIT", "MCS", "SCS", "TPR"))
for i in range(len(dorms)):
    print("%11s %03s %03s (%6.2f%% full) --> majors: [%02s, %02s, %02s, %02s, %02s, %02s]" %
            (names[i], sum(dorms[i]), capacities[i], 100 * sum(dorms[i]) / capacities[i],
            dorms[i][0], dorms[i][1], dorms[i][2], dorms[i][3], dorms[i][4], dorms[i][5]))
```