Carnegie Mellon University
21-393 Operations Research II
December 21^{st}*, 2011*

Dormitory Assignment Optimization

Shrui Hu
Adeline Li
Chang Qi
Kenneth Wong

Abstract

This paper describes our final project for the Operations Research II Course in Carnegie Mellon University, we decided to investigate the topic of "dormitory assignments". All the data we have are from the Housing and Dining Services for freshman dormitory assignments for Fall 2011. The final results from this research was that our proposed assignment algorithm was better than the current one.

We would love to thank professor Alan Frieze for his feedback, and the Housing and Dining Services for the support in providing us with the data.

Introduction

Each year, the housing office faces the problem of assigning every incoming freshman to a dormitory room. With over 1,500 freshmen to allocate and possibly insufficient number of rooms, Housing and Dining Services tries to satisfy every students requests as much as possible. However, along with the growing student body each year and the many external preferential factors influencing the decisions of each student, the assignment process becomes much more difficult.

Because all incoming freshmen are required to live on campus, the Housing and Dining Services must assign all the freshmen to a specific dorm. To make the experience more enjoyable, the freshmen are asked to list out personal preferences and Housing and Dining Services tries to allocate each freshmen to a dorm while satisfying as many students preferences as possible.

Problem

This is where our research project comes in. Our research aims to use the techniques we learned in class to find the best possible student-dorm assignments such that the combined satisfaction of all students is maximized. By solving this problem, we hope to maximize the number of students obtaining a room of their choice and achieve a more efficient algorithm for this process.

The maximization of total student satisfaction is important because it could impact overall mood on campus. A student satisfied with their dorm assignment would probably be more likely to start their day off with a better mood and outlook. Another crucial role dorm satisfaction takes is that it can determine the number of students who choose to stay with on-campus housing the next year. Since a part of Carnegie Mellons revenue comes from dormitory fees, maximizing students satisfaction can possibly maximize CMUs revenue from dorms.

Current Method

Before the start of a new school year, every incoming freshman is asked to fill out an online application asking them to list their top choices for dorm preferences. The Housing and Dining Services at Carnegie Mellon University also asks other for other personal preferences such as specific roommate requests, dorm gender, noise level, interests, and sleeping habits. After all the information is submitted, hard copies of each application are printed out and then organized by gender, school and major, and by deposit submission

date. Applications are then looked at and considered in the order that the deposit was received. Students who submitted their deposits earlier are more likely to have all their preferences satisfied. An exception to this roommate requests, which is held in highest priority after deposit date. If Student A and Student B requested to be roommates, and Student A submits her deposit early but Student B submits her deposit later Student B will still be considered at the same time as Student A because they requested each other as roommates. Though Student A and B may not get their desired room, they are guaranteed a room together. This process continues until every student is assigned a dorm.

According to the data provided by the Housing and Dining Service, each enrolled freshman is asked to submit their dorm preference form. They need to provide their top 5 dorms in order as well as to rank the five room types. (Single, Double, Triple, Quad and Apartment) Each form is separated by gender and then ordered by deposit date. The person who paid reservation deposit earlier will be in a higher order.

Proposed Method

In order to improve the method for the current assignment, we implement an optimization algorithm to try and improve the satisfaction of the whole freshmen population.

The algorithm we implemented to our assignment problem is the Hungarian algorithm. We had picked this because unlike the current method, it is not greedy and can give us a globally optimal solution.

The following is how the algorithm works:

1. Arrange your information in a matrix such that your variables are the along the top and the left side of the matrix and each element is the cost for each pair of variables
2. Check to see if your matrix is a square matrix. If it isnt, then add dummy rows and columns such that each dummy row/column is the same as the largest number in the matrix.
3. Find the smallest value of each row and subtract that value from every element in that row. This reduces the rows of the matrix.
4. Reduce the columns by finding the smallest element of each column and subtracting that value from every element in that row.
5. Draw the smallest number of horizontal and vertical lines so that the zero elements are crossed out. If the number of lines is equal to the number of rows, go directly to step 9.

6. Find the smallest element that isnt crossed out and add that value to every element that is covered. Add the value twice if the element is crossed out twice.
7. Find the smallest element in the matrix and subtract it to every element.
8. Cross out the zeroes with the minimum number of lines again and if the number of lines isnt equal to the number of rows, go back to step 6.
9. Choose a set of zeroes such that each row and column has only one zero selected.
10. Comparing to the original matrix, find the values corresponding to the positions of the zeroes you selected in Step 9 (disregarding dummy rows and columns). These values summed should give the total minimum cost.

Assumptions

In order to do this project, we have made several assumptions:

1. There are enough rooms to house every incoming freshman (after all upperclassmen had received their assignments).
2. Every student is treated equally. This means that no student will have special advantages over another (with the exception of deposit submission date) in terms of satisfaction with his or her dorm assignment

Constraints

We will have a set of hard constraints and a set of soft ones. Hard constraints are ones that should never be broken. For example, each student sleeps on exactly one bed, the number of students assigned to a room must equal the total number of beds in that room, and that all available spaces are used. Soft constraints, on the other hand, are constraints that are allowed to be broken only out of necessity. For example, soft constraints could be preferential factors such as proximity to campus, co-ed living environment, room facilities, and prices.

Cost Allocation

Our cost assignment works as follows:

Room Type	Assignments
1st Choice	+0 Cost
2nd Choice	+5 Cost
3rd Choice	+10 Cost
4th Choice	+15 Cost
5th Choice	+20 Cost

Dormitory Assignments	Cost
1st Choice	+0 Cost
2nd Choice	+1 Cost
3rd Choice	+2 Cost
4th Choice	+3 Cost
5th Choice	+4 Cost
No Choices Met	+25 Cost

Other Cost Assignments	Cost
Male Assigned to Female Only Dorms	+M Cost
Female Assigned to Male Only Dorms	+M Cost

Using this assignment function, we pair the m students with the n dorm room choices, getting assignment matrix of dimensions m by n. The vertical axis will be the students, while the horizontal axis will be the dorm rooms. So there will be a cost for each combination of assigning each of student j to each of dorm room i. In this context, the cost actually means how unsatisfied a student would be with this assignment, so while we are trying to maximize the satisfaction of all students, we are also trying to minimize the dissatisfaction of each student.

Dormitory Assignment

Given that we have so much data, it is impossible to do this by hand. We used a matlab code that reproduces the Hungarian algorithm to do the assignment for us. However, for demonstration purposes, we did an example with a random sample of 10 students and 10 dorm rooms below.

Notice we may have the possibility of assigning a make and female together, for example. If a group of rooms consists of 5 doubles, say there were 5 males and 5 females assigned to this group of rooms, one room would house a male and female. We would have to remedy this after the assignment, it would make the assignment less optimal, but the problem would be fixed. We could not have prevented this problem from the beginning, since we would not the proportion of males and females assigned to each dorm room, and altering the assignment would make it greedy and defeat the purpose of optimizing the whole freshman population.

Demonstration

Lets choose students from the real data and see if our method is working well. We Choose the first ten students from the data. Now suppose the rooms we have include a Donner Double, a Hamerschlag Single, a Mudge Triple, an E-Tower Double, a Stever Single and a Donner Single. The data for these 10 students are shown below:

	Gender (Female = 0, Male = 1)										
1	1	D	S	T	Q	A	Mudge	Donner	Hamerschlag	E-Tower	Scobell
2	0	D	T	S	Q	A	E-Tower	Stever	Mudge	Gardens	West
3	1	D	T	S	Q	A	Mudge	Stever	E-Tower	Donner	Residence
4	0	D	T	S	Q	A	Donner	McGill	Mudge	E-Tower	Residence
5	1	D	A	S	T	Q	Mudge	Stever	E-Tower	Webster	West
6	1	S	D	A	T	Q	Mudge	E-Tower	Stever	Donner	Residence
7	1	D	S	T	Q	A	E-Tower	Mudge	Stever	Donner	Residence
8	1	D	T	S	Q	A	Stever	Mudge	Donner	E-Tower	Gardens
9	1	D	T	S	Q	A	Stever	E-Tower	Mudge	Donner	Resnik
10	1	D	T	S	Q	A	Stever	E-Tower	Mudge	Donner	Resnik

We still use the greedy method first to solve the problem. Just as what we did in the previous example, we will assign student 1 to Donner Double, 2 to E-Tower Double, 3 to Donner Double, 4 to E-Tower Double, 5 to Mudge Triple, 6 to Stever Single, 7 to Mudge Triple, 8 to Mudge Triple, 9 to Donner Single and 10 to Hamerschlag Single. The result is shown below.

Current Method
1 Donner Double
2 E-Tower Double
3 Donner Double
4 E-Tower Double
5 Mudge Triple
6 Stever Single
7 Mudge Triple
8 Mudge Triple
9 Donner Single
10 Hamerschlag Single

The total cost for this method is calculated to be 88. Then we apply our method to this problem. The assignment problem matrix will be:

		Do D	Do D	Ha S	Mu T	Mu T	Mu T	E D	E D	St S	Do S
M	A	1	1	7	10	10	10	3	3	30	6
F	B	25	25	M	7	7	7	0	0	11	35
M	C	3	3	35	5	5	5	2	2	12	13
F	D	0	0	M	7	7	7	3	3	35	10
M	E	25	25	35	15	15	15	2	2	6	35
M	F	8	8	25	15	15	15	6	6	2	3
M	G	3	3	30	11	11	11	0	0	7	8
M	H	2	2	35	6	6	6	3	3	10	12
M	I	3	3	35	7	7	7	1	1	10	13
M	J	3	3	35	7	7	7	1	1	10	13

Entry (m,n) means the cost or how unsatisfied the student m will feel if we assign him into bed n. In this matrix we see some Ms because girls cannot be assigned to Hamerschlag. Since this is a 10 by 10 matrix. Its really hard to solve it by hand. So we use a MATLAB code of Hungarian Method uploaded online for free to solve it. (The code is provided in the reference page) The assignment result to put student

1 to Hamerschlag Single, 2 to E-Tower Double, 3 to Mudge Triple, 4 to Donner Double, 5 to Stever Single, 6 to Donner Single, 7 to E-Tower Double, 8 to Mudge Triple, 9 to Mudge Triple and 10 to Donner Double. Then we check the gender and find that student 2 (female) and student 7 (male) are in the same room. Student 4 (female) and student 10 (male) are also in the same double room. So we need to switch two of the four people. If we switch student 2 to Donner Double, 25 will be added to the cost. If we switch student 4 to E-Tower Double, 3 will be added to the cost. Similarly, if we switch student 7 to Donner or student 10 to E-Tower, 3 or -2 will be added. So we decide to switch student 4 and student 7 to make the cost smallest. The final result is shown below:

Proposed Method
1 Hamerschlag Single
2 E-Tower Double
3 Mudge Triple
4 E-Tower Double
5 Stever Single
6 Donner Single
7 Donner Double
8 Mudge Triple
9 Mudge Triple
10 Donner Double

The total cost then is calculated to be 43 which is about half of the cost of the current method. This example shows that with more data, our method works even better by successfully decreasing the cost.

Results

From our online survey, the estimated current freshman dorm satisfaction level is 82%. In another word, out of the 130 CMU students who completed the survey, 106 of them are generally satisfied with their freshman dorm. This high current satisfaction level was unexpected , and leaves us with very little space for improvements.

Comparison

Our new system has several advantages compare to the current system. First of all, from our test run results, we know that more freshman will be satisfied with their dorms using our proposed dorm selection process. For both cases considering and not considering gender, our method significantly lowered the cost, ie level of satisfaction, by nearly half. Second, the algorithm we used is not a greedy algorithm, which

eliminates the possibility that the optimal solution is only a locally optimal solution that approximates a global optimal solution. Also, it is easier to implement new variables in our algorithm. To do so, we simply add a variable to our cost assignment function and solve for the optimal solution under the new function. Moreover, with simple explanation, our algorithm could be understood by people with little Operations Research background. This would allow upcoming freshman to have a basic understanding of the essences of the room assignment process. On the other hand, there are also disadvantages of our algorithm. First, since we do not take into account of the order of payment, it might be unfair to those who paid first. In the current system, the first person to make the tuition deposit has the highest priority in the room assignment process, and so on, which we think is reasonable. However, it was problematic to include this factor in our algorithm, so we left it to future discussion. Second, there is a trade-off between student satisfaction and algorithm efficiency. Although our process significantly increase student satisfaction, it also increases run time to $O(n^3)$. Also, since we need to gather all the data before we start running our algorithm, it is impossible for us to adopt a rolling process, which means that there will be a long lead time and students will have to wait for longer. Lastly, unlike the current process, our algorithm does not guarantee required roommates. This would be something to work on in future expansions.

Expansion

There are some future expansions that we have in mind. We were not able to integrate these expansions into our algorithm due to time and knowledge constraints, and given the chance we would like to include these expansions in our project. First of all, some of the data in the current system were not provided to us by the Housing office for confidentiality purposes, such as quiet floor requests and dorm distance preferences of students. These could potentially be very important factors to students overall satisfaction level of dorms. For example, being on the quiet floor might be very important to some students, therefor not considering quiet floor requests might lead to potential high cost, or satisfaction level. We would also like to add in roommate requests to our algorithm. On the other hand, there are some factors no included the current system that we consider to be crucial. For example, some students might prefer dorms with dinning. Also, being close to certain facilities could affect students preference for dorms as well. We could add in these factors by adding new variables in cost assignment function. However, since some of these factors are not conventional factors, some modification might be required.

Future Application

Dorm assignment problem could have a wide range of applications. Some similar problems could be solved by directly applying our algorithm. Some examples of such type of problems are assigning professors to classes problem, assigning cars to parking spaces problem, and assigning students to project problems. By replacing our cost variables with new relevant cost variables, these problems could be solved using our process. There are also some assignment problems that could be solved with a modified version of our algorithm. One example would be the product pricing problem. Retailers might want to sell products at different prices to different group of consumers (membership) or in different countries depending on their buying powers. In this case, we can not directly apply our algorithm because we are not simply assigning a set of prices to a set of consumers. Instead, we need to include in our algorithm a function that determines the prices first given the cost variables and then assign these prices to the appropriate countries. Some other problems that could be potentially solved or estimated by a modified version of our algorithm are

- Dorm Mate Assignment problem (different from roommate requests factor)
- Assignment problems involving time
- Minimizing Transport Distance problem

Conclusion

In conclusion, from test results and process analysis, we believe that our algorithm is a improved version of room assignment process. We greatly improved student satisfaction level using our algorithm. However, given the high current satisfaction level, there is only limited spaces for further improvement. Given a chance, we would like to improve our algorithm even more by adding more relevant cost variables. Also, even our process is an improved version of current process, depends on the cost for implementation if might not be worth it to change the current system given the high current satisfaction level.

Appendix

References

Matlab Code for the Hungarian Method:

http://www.mathworks.com/matlabcentral/fileexchange/11609

A Summary of the Hungarian Method

http://canmedia.mcgrawhill.ca/college/olcsupport/stevenson/om3ce/IOM_applets/hungarian Method/Hungarian.htm

Code

```
function [Matching,Cost] = Hungarian(Perf)
%
% [MATCHING,COST] = Hungarian_New(WEIGHTS)
%
% A function for finding a minimum edge weight matching given a MxN Edge
% weight matrix WEIGHTS using the Hungarian Algorithm.
%
% An edge weight of Inf indicates that the pair of vertices given by its
% position have no adjacent edge.
%
% MATCHING return a MxN matrix with ones in the place of the matchings and
% zeros elsewhere.
%
% COST returns the cost of the minimum matching

% Written by: Alex Melin 30 June 2006


 % Initialize Variables
 Matching = zeros(size(Perf));

% Condense the Performance Matrix by removing any unconnected vertices to
% increase the speed of the algorithm

  % Find the number in each column that are connected
    num_y = sum(~isinf(Perf),1);
  % Find the number in each row that are connected
    num_x = sum(~isinf(Perf),2);

  % Find the columns(vertices) and rows(vertices) that are isolated
```

```
    x_con = find(num_x~=0);
    y_con = find(num_y~=0);

  % Assemble Condensed Performance Matrix
    P_size = max(length(x_con),length(y_con));
    P_cond = zeros(P_size);
    P_cond(1:length(x_con),1:length(y_con)) = Perf(x_con,y_con);
    if isempty(P_cond)
      Cost = 0;
      return
    end

    % Ensure that a perfect matching exists
      % Calculate a form of the Edge Matrix
      Edge = P_cond;
      Edge(P_cond~=Inf) = 0;
      % Find the deficiency(CNUM) in the Edge Matrix
      cnum = min_line_cover(Edge);

      % Project additional vertices and edges so that a perfect matching
      % exists
      Pmax = max(max(P_cond(P_cond~=Inf)));
      P_size = length(P_cond)+cnum;
      P_cond = ones(P_size)*Pmax;
      P_cond(1:length(x_con),1:length(y_con)) = Perf(x_con,y_con);

%************************************************
% MAIN PROGRAM: CONTROLS WHICH STEP IS EXECUTED
%************************************************
  exit_flag = 1;
  stepnum = 1;
  while exit_flag
    switch stepnum
```

```
      case 1
        [P_cond,stepnum] = step1(P_cond);
      case 2
        [r_cov,c_cov,M,stepnum] = step2(P_cond);
      case 3
        [c_cov,stepnum] = step3(M,P_size);
      case 4
        [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(P_cond,r_cov,c_cov,M);
      case 5
        [M,r_cov,c_cov,stepnum] = step5(M,Z_r,Z_c,r_cov,c_cov);
      case 6
        [P_cond,stepnum] = step6(P_cond,r_cov,c_cov);
      case 7
        exit_flag = 0;
    end
  end

% Remove all the virtual satellites and targets and uncondense the
% Matching to the size of the original performance matrix.
Matching(x_con,y_con) = M(1:length(x_con),1:length(y_con));
Cost = sum(sum(Perf(Matching==1)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   STEP 1: Find the smallest number of zeros in each row
%           and subtract that minimum from its row
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [P_cond,stepnum] = step1(P_cond)

  P_size = length(P_cond);

  % Loop throught each row
  for ii = 1:P_size
```

```
    rmin = min(P_cond(ii,:));
    P_cond(ii,:) = P_cond(ii,:)-rmin;
  end

  stepnum = 2;

%**************************************************************************
%   STEP 2: Find a zero in P_cond. If there are no starred zeros in its
%           column or row start the zero. Repeat for each zero
%**************************************************************************

function [r_cov,c_cov,M,stepnum] = step2(P_cond)

% Define variables
  P_size = length(P_cond);
  r_cov = zeros(P_size,1);  % A vector that shows if a row is covered
  c_cov = zeros(P_size,1);  % A vector that shows if a column is covered
  M = zeros(P_size);        % A mask that shows if a position is starred or primed

  for ii = 1:P_size
    for jj = 1:P_size
      if P_cond(ii,jj) == 0 && r_cov(ii) == 0 && c_cov(jj) == 0
        M(ii,jj) = 1;
        r_cov(ii) = 1;
        c_cov(jj) = 1;
      end
    end
  end

% Re-initialize the cover vectors
  r_cov = zeros(P_size,1);  % A vector that shows if a row is covered
  c_cov = zeros(P_size,1);  % A vector that shows if a column is covered
  stepnum = 3;
```

```
%*************************************************************************
%   STEP 3: Cover each column with a starred zero. If all the columns are
%           covered then the matching is maximum
%*************************************************************************

function [c_cov,stepnum] = step3(M,P_size)

  c_cov = sum(M,1);
  if sum(c_cov) == P_size
    stepnum = 7;
  else
    stepnum = 4;
  end

%*************************************************************************
%   STEP 4: Find a noncovered zero and prime it.  If there is no starred
%           zero in the row containing this primed zero, Go to Step 5.
%           Otherwise, cover this row and uncover the column containing
%           the starred zero. Continue in this manner until there are no
%           uncovered zeros left. Save the smallest uncovered value and
%           Go to Step 6.
%*************************************************************************
function [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(P_cond,r_cov,c_cov,M)

P_size = length(P_cond);

zflag = 1;
while zflag
    % Find the first uncovered zero
      row = 0; col = 0; exit_flag = 1;
      ii = 1; jj = 1;
      while exit_flag
```

```
          if P_cond(ii,jj) == 0 && r_cov(ii) == 0 && c_cov(jj) == 0
            row = ii;
            col = jj;
            exit_flag = 0;
          end
          jj = jj + 1;
          if jj > P_size; jj = 1; ii = ii+1; end
          if ii > P_size; exit_flag = 0; end
      end

    % If there are no uncovered zeros go to step 6
      if row == 0
        stepnum = 6;
        zflag = 0;
        Z_r = 0;
        Z_c = 0;
      else
        % Prime the uncovered zero
        M(row,col) = 2;
        % If there is a starred zero in that row
        % Cover the row and uncover the column containing the zero
          if sum(find(M(row,:)==1)) ~= 0
            r_cov(row) = 1;
            zcol = find(M(row,:)==1);
            c_cov(zcol) = 0;
          else
            stepnum = 5;
            zflag = 0;
            Z_r = row;
            Z_c = col;
          end
      end
end
```

```
%*************************************************************************
% STEP 5: Construct a series of alternating primed and starred zeros as
%         follows.  Let Z0 represent the uncovered primed zero found in Step 4.
%         Let Z1 denote the starred zero in the column of Z0 (if any).
%         Let Z2 denote the primed zero in the row of Z1 (there will always
%         be one).  Continue until the series terminates at a primed zero
%         that has no starred zero in its column.  Unstar each starred
%         zero of the series, star each primed zero of the series, erase
%         all primes and uncover every line in the matrix.  Return to Step 3.
%*************************************************************************

function [M,r_cov,c_cov,stepnum] = step5(M,Z_r,Z_c,r_cov,c_cov)

  zflag = 1;
  ii = 1;
  while zflag
    % Find the index number of the starred zero in the column
    rindex = find(M(:,Z_c(ii))==1);
    if rindex > 0
      % Save the starred zero
      ii = ii+1;
      % Save the row of the starred zero
      Z_r(ii,1) = rindex;
      % The column of the starred zero is the same as the column of the
      % primed zero
      Z_c(ii,1) = Z_c(ii-1);
    else
      zflag = 0;
    end

    % Continue if there is a starred zero in the column of the primed zero
    if zflag == 1;
```

```
      % Find the column of the primed zero in the last starred zeros row
      cindex = find(M(Z_r(ii),:)==2);
      ii = ii+1;
      Z_r(ii,1) = Z_r(ii-1);
      Z_c(ii,1) = cindex;
    end
  end

  % UNSTAR all the starred zeros in the path and STAR all primed zeros
  for ii = 1:length(Z_r)
    if M(Z_r(ii),Z_c(ii)) == 1
      M(Z_r(ii),Z_c(ii)) = 0;
    else
      M(Z_r(ii),Z_c(ii)) = 1;
    end
  end

  % Clear the covers
  r_cov = r_cov.*0;
  c_cov = c_cov.*0;

  % Remove all the primes
  M(M==2) = 0;

stepnum = 3;

% *************************************************************************
% STEP 6: Add the minimum uncovered value to every element of each covered
%         row, and subtract it from every element of each uncovered column.
%         Return to Step 4 without altering any stars, primes, or covered lines.
%**************************************************************************

function [P_cond,stepnum] = step6(P_cond,r_cov,c_cov)
```

```
a = find(r_cov == 0);
b = find(c_cov == 0);
minval = min(min(P_cond(a,b)));

P_cond(find(r_cov == 1),:) = P_cond(find(r_cov == 1),:) + minval;
P_cond(:,find(c_cov == 0)) = P_cond(:,find(c_cov == 0)) - minval;

stepnum = 4;

function cnum = min_line_cover(Edge)

  % Step 2
    [r_cov,c_cov,M,stepnum] = step2(Edge);
  % Step 3
    [c_cov,stepnum] = step3(M,length(Edge));
  % Step 4
    [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(Edge,r_cov,c_cov,M);
  % Calculate the deficiency
    cnum = length(Edge)-sum(r_cov)-sum(c_cov);
```