Dormitory Lottery Solutions

Ryan Williams, 6.034 Project 1

November 6, 2002
Abstract

The author has applied Dimitri Bertsekas's Auction algorithm to achieve an optimal solution to MIT's dormitory lottery problem. The goal of the dormitory lottery is to assign the students to dorms so that overall their preferences are maximized, a challenging problem given the number of possible permutations of students and dorms. This implementation also takes into account the gender balances desired for each dormitory, making it useful in real-life situations. The algorithm runs in $O(n^2)$ time, which in practice takes less than a minute for one thousand students.
The Problem

MIT’s housing system rests on the shoulders of a good lottery. Students rank dorms in order of their desires, and the lottery attempts to reconcile all their desires in an optimal way. The problem is one of allocating scarce resources (space in the dorms) in such a way that constraints and preferences are respected.

The goal of the system is to maximize the preferences of the aggregate student body. It is not hard to find a solution that “works” - the challenge is rather to find a solution that optimizes all factors.

Participants in the lottery include all undergraduates living in on-campus housing, about 3000 students. When the lottery starts, we assume that the participants are already assigned to dormitories. Each student can then choose either to rank $k$ dormitories or to “squat”, i.e. stay where they are. $k$ can vary per-student.

1.1 Rules and Constraints

The rules are:

- Students can live in one of eleven dormitories.

- Each dormitory has a finite amount of beds.

- If the system cannot find a place for a student in their $k$ preferences, they are left in their current assignment.

- Students squatting either through choice or due to the previous constraint do not change dorms from their original assignment.

- Male/female ratio must be maintained. Each dorm has a set of male and female slots, which must be respected.

Being bound by these constraints, our goal is to optimize the following factors:

- Student preferences

- Ratio of underclassmen to upperclassmen

- Seniority

In general, we can expect that most of the participants in the lottery will choose to squat. This factor will tend to significantly reduce the size of the solution space and make following the constraints more important.
1.2 Simplifying Assumptions

The above description does not give credit to the complexity of the problem. In addition to those factors, there are also the problems related to Residence-Based Advising (RBA), and the cultural houses. Since RBA is a binding commitment, and cultural houses "extend bids", these are handled by human assignment, and therefore my system must be tolerant of already-assigned people.

Further, I will assume that the number of students needing rooms is exactly equal to the number of slots available for them. The male-female balance is addressed by giving each dormitory a number representing "male slots" and "female slots". It is up to the operator to ensure that these match the input data, and we will assume that the number of males and females in the students to be assigned are equal to the number of corresponding slots.

1.3 Challenges

One of the particular difficulties of this problem type is that often there are times when there are many local maximal optimizations that the system may "settle" into.

One such situation is presented in Figure 1.1. If there were a situation where the assignments \{(1,5), (2,4), (3,6)\} were made, a naive algorithm might not be able to achieve the optimum assignment \{(1,4), (2,6), (3,5)\} since it is not possible to exchange the assignments of any pair of students without decreasing the optimality of the solution.

![Diagram](image)

Figure 1.1: An illustration of a potential cycle. The students are the nodes labeled 1-3, the dormitories the nodes labeled 4-6.
Solution

Initially I considered constraints-based solutions, ones that were basically a method of constraining the possible assignments, and then searching through the possible assignments for an optimal solution. Given about 1000 students and 10 dorms, this would imply a solution space of $1000^0$, or $10^{02}$, a dauntingly high number to search. Constraining the problem could reduce this problem. As described in lecture, the constraints reduction would take about $1000^3 = 10^9$ time. If we assume that the problem is constrained enough that each student's choices are reduced to about 2, the search would then cover around one million possibilities. This could be solved in a reasonable time if they could be generated quickly, but each would take a time proportional to the number of students involved, bringing the number of operations up to one billion in total, an $n^3$ operation. However, I was more worried about the issue highlighted in Figure 2.2, that the search, unless it were backtracking, would pick a wrong route.

![Diagram](image)

Figure 2.2: Possible error in a non-thorough algorithm.

So I decided to attempt to find solutions that are less direct and more aimed at achieving optimality from the start.

This problem is not a new one; the field of network optimization labels this the “assignment problem”, a specific case of the minimum cost flow problem [JB80]. The assignment problem is one in which you have a set $N_1$ of nodes (students) that must be connected via arcs to elements of the node set...
$N_2$ (dorms). This is a much simpler problem than the general minimum-cost flow problem because the minimum flows are considered to be zero, capacity is irrelevant, and all flows are unit flows. A graphical illustration with five students and three dormitories is shown in Figure 2.3.

Figure 2.3: The network representation of student preferences for dorms. The numbers on the arcs represent both the student preference and the flow cost associated with an assignment.

Finding an assignment of dormitories to students so that the sum of the arc costs is minimized is the solution to our problem. You can see that the set $\{(2,8), (1,6), (3,7), (4,8), (5,7)\}$ is the optimal solution for the example in Figure 2.3, if we assume that the dormitories have infinite capacity. Since the dormitories in fact have a finite capacity, we will have to modify our problem definition somewhat.

In light of the fact that the assignment problem applies to one-on-one associations, and we wish to apply it to a many-on-one problem, we will change the representation of the problem so that it fits the one-on-one model. This is done by making each dormitory of capacity $s$ be represented by $s$ “slots” that each can carry one student. Our example from before, Figure 2.3, is now expanded to show that dorms 6 and 7 each have capacities of 2.

Now we have, through transformation, a problem in a simple, symmetric form, ready for application of an algorithm.
2.4 Auction Algorithm

After some review, I decided that the auction algorithm [Ber92], developed by Prof. Bertsekas in 1989, seemed to be most appropriate way to solve it. The algorithm is $O(n^2)$ in its simplest form, and there are a variety of ways to cut it down to an $O(n \log n)$ solution (these are average-case times). This is a fair asymptotic improvement in speed over the search method, which took $O(n^3)$.

In order to speed up the algorithm somewhat, I used the concept of object similarity, discussed by by [Ber92]. This is not a “pure” auction algorithm, and therefore may be a bit harder to understand. For a better conceptual understanding of the auction algorithm, a reading of the excellent [Ber92] is best.

The auction algorithm comes up with an optimal solution by allowing students to “bid” on dorm slots. Each student $i$ is assigned a benefit, $b_{ij}$ relative to a particular dorm $j$. Each dorm $j$ keeps track of a price, $p_j$, and a set of slots $D_j$. Each slot $g$ keeps track of a contention threshold, $\hat{p}_g$. The price $p_j$ of each dorm $j$ is the minimum contention threshold of all the slots in that dorm:

$$p_j = \min_{x \in D_j} \hat{p}_x$$  \hspace{1cm} (2.1)

The algorithm iterates over the students in two phases.

**Bidding phase:**

During this phase, all the unassigned students bid against the dorm slots. Each student $i$ bids against the best slot as far as that student
Table 2.1: Useful terms

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>( i )</td>
<td>( a_{ij} ) How much the student wants ( j ).</td>
</tr>
<tr>
<td>Dorm</td>
<td>( j )</td>
<td>( p_j ) Cost of choosing a dorm.</td>
</tr>
<tr>
<td>Slot</td>
<td>( g )</td>
<td>( \hat{p}_g ) Cost of a particular slot.</td>
</tr>
<tr>
<td>Benefit</td>
<td>( a_{ij} )</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>( p_j )</td>
<td></td>
</tr>
<tr>
<td>Contention Threshold</td>
<td>( \hat{p}_g )</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>( v_i )</td>
<td>The value for the student's best choice.</td>
</tr>
<tr>
<td>Value 2</td>
<td>( w_i )</td>
<td>The value for a student's second-best choice.</td>
</tr>
<tr>
<td>Bid Increment</td>
<td>( \gamma_i )</td>
<td>A student's bid increment.</td>
</tr>
<tr>
<td>Tolerance</td>
<td>( \epsilon )</td>
<td>Constant of accuracy (and speed).</td>
</tr>
</tbody>
</table>

is concerned, \( g_i \), which is in the best dorm, \( j_i \):

\[
g_i \text{ such that } a_{ij} - \hat{p}_g \geq \max\{a_{ij} - \hat{p}_g\} - \epsilon \tag{2.2}
\]

That is, a "value", \( v_i \), which is equal to the difference between the current contention threshold of the dorm slot and the benefit to the student of the dorm \( (a_{ij} - \hat{p}_g) \), is computed for each slot \( g \) that the student is considering.

The student then calculates the value of their first and next-best-outside-of-the-dorm choice,

\[
v_i = \max\{a_{ij} - \hat{p}_g\} \tag{2.3}
\]

\[
w_i = \max_{g \in D \setminus \{i\}} \{a_{ij} - \hat{p}_g\} \tag{2.4}
\]

This is used to calculate the bid increment of the student, \( \gamma_i \),

\[
\gamma_i = v_i - w_i + \epsilon \tag{2.5}
\]

**Assignment phase:**

Every dorm slot bid on in the previous phase selects its highest bidder, \( i_g \),

\[
i_g = \arg \max \gamma_i \tag{2.6}
\]

and sets its contention threshold by the addition of the price and the bid increment of the selected student, \( \max \gamma_i \):

\[
\hat{p}_j = p_j + \max \gamma_i \tag{2.7}
\]

The high-bidding student \( i_j \) is then assigned to the dorm slot \( j \), and any student previously holding that slot becomes unassigned. After all assignments are made, the prices for each dorm are reset to the minimum contention threshold, as described by Equation 2.1.
One can see that this algorithm keeps two sets of students, assigned and unassigned, and the algorithm terminates when there are no students in the bidding phase, or equivalently when all students are assigned to some slot. I leave the proof of correctness of this algorithm to Prof. Bertsekas in Appendix 2 of [Ber92].

2.5 Assigning Benefits

The choice of a function that maps dorm preferences (which are maximally "desirable" when they are lowest, e.g. first place is best) to a "benefit" that increases with desirability, is not as easy as it might seem. In fact, with the auction algorithm as the backdrop, assignment of the benefits becomes the overarching concern. Almost any desired bias or special effect can be achieved by tweaking the benefits.

The most straightforward approach is to define some global value $k$ and to subtract the student’s preference ranking of a dormitory to achieve an $a_{ij}$ that changes linearly with preference. My initial implementation uses this approach, leading to good results. However, one aspect of this system is that it is equally optimal for one person to get their 5th choice while 5 others get their first as it is for that one person to get their first choice and 5 others get their second choice.

Another approach would be to define some non-linear function to scale the preferences in some way. One conceivable approach would be to assign the first-ranked dorms a much higher benefit than the lower-ranked dorms, in an effort to maximize the number of top-choice dorms, and minimize the number of people getting their last choice. Any such approach must be careful to keep the range of values equivalent for students regardless of how many dorms they rank.

Given that we have $k$, the maximum possible ranking of a dorm, we can calculate a "squared" algorithm to achieve nonlinearity:

$$a_{ij} = k - \left\lfloor k(p_{ij}/k)^2 \right\rfloor \quad (2.8)$$

This resulted in a mapping of values as per Figure 2.5. This is close to what I

![Figure 2.5: Mapping 1](image-url)
intended. However, the first four preferences are lumped into the same benefit, and steps had to be taken to distinguish them. I changed the formula to:

$$a_{ij} = k^2 - p_{ij}^2$$

(2.9)

Which results in the more distinct distribution seen in Figure 2.6. This seems to have the effect desired, in that lower preferences are assigned increasingly lower benefits. With this model, it is unlikely that someone will get below their third choice. Note that it doesn’t matter how high the baseline benefit is.

2.6 Male/Female Slots

The program will deal with the problem of male and female slots by running two disjoint auctions simultaneously, one for male students, one for female students. Each dorm will be split into two “dorms”, one representing each gender’s slots. The students’ benefits will be assigned relative to the correspondingly gendered dorm. Any student’s preferences will therefore be meaningful for only half of the dorms. My implementation assigns a preference of 1000 to the incorrectly gendered dorms, to discourage assignment and to make outliers clear.

Initially I had considered running two separate instances of the auction algorithm, and that may have been more transparent in the code. However, I wanted to emphasize that almost any result can be achieved by using the “standard” auction and varying the preferences.

2.7 Feasibility

Infeasibility occurs when there is no assignment of students to slots that will maintain a one-to-one correspondence. Figure 2.7 is a trivial infeasible situation. Note that infeasibility has very little to do with nonoptimality. [Ber92] describes several ways to detect infeasibility. One of these ways involves adding artificial pairs that have very low benefits and do not play a part in the auction unless the situation is infeasible. I reasoned that no situation could be infeasible that had
the same number of students as slots, where each student has some preference for each slot. In this situation, an infeasible assignment is when a student is assigned to a dorm for which he or she has a very low benefit. The threshold low benefit is \(-2n - 1\). In the process of assigning benefits to students for dorms, each student is assigned an extremely low benefit for each dorm that they did not specify a preference.

The end result is that each student has some benefit for each dorm, and infeasibility is detected when it happens.

2.8 Future Work

In implementing the auction algorithm, I included the possibility of implementing \(\epsilon\)-scaling, which is a technique for improving the speed by starting \(\epsilon\) at a high value, which leads to a quick if nonoptimal solution, then slowly reducing its value to 1 over a series of runs. Each run would be closer to the solution, and the final urn with \(\epsilon\) equal to 1 would be the optimal run. This is a speed technique, and results in faster but not more optimal solutions, so I declined to implement it since the algorithm seems fast enough already.

I also almost added support for caching the students' top choices. This method assumes that a student's top choices don't change too often, and therefore stores their top three. A simple check determines whether or not the top three are still the top three for each student, obviating the need for the \(O(n)\) first- and second-choice checks required for each student. Again, I did not implement this (though the data structures are there) because it seemed superfluous.

Other speed increases are also possible, though at the cost of major modification to the engine. One such modification would be to run the auction both forwards and backwards.

Further work could be done to make the application more "user-friendly". Currently the input files are hardcoded into the main program file, the program relies heavily upon the data being correct, and makes several assumptions about the way preferences are entered. A more robust implementation would ensure that the number of students and slots is equal (or gets around the discrepancy somehow), and allows one to specify preferences out-of-order in the input file.

Lastly, I noted that there are many equally-optimal solutions for a given input. Currently the program operates completely deterministically, producing exactly the same assignment each time it's run. While this is nice for analysis,
it would be better in the interest of fairness to introduce randomness so that "alphabet favoritism" is not exhibited. A good way to do that would be to modify the function that generates the list of unassigned students so that it randomly permutes the list each time it runs.
Results

I made many test runs of the algorithm to see if it worked, moving upwards in complexity as I went. I first tested the algorithm with a trivial case, where the correct result is obvious. In this mapping, students are the rows, the dorms are the columns, and the values are the “benefit” associated for that person and dorm:

<table>
<thead>
<tr>
<th></th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

This produced the output:

Students:
Name   Assignment
A      J
B      L
C      K

as expected.

A more complex case:

<table>
<thead>
<tr>
<th>J(2)</th>
<th>K(1)</th>
<th>L(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

produced the following result:

Assignments:
Student Dorm Benefit
A J 3
B J 3
C L 2
D K 4
E L 2

Every student in this toy scenario is given its highest-valued choice.

I was satisfied that the algorithm was producing the correct results, and discovered that coming up with more complex test cases was very challenging.

3.9 Run 1

I decided to try it on a data set that came from this summer’s lottery. In order to simplify things, I dropped all the information on gender, special assignments,
and "special" living groups like cultural houses. This left 854 nameless students and their preferences for 10 dorms. I converted each preference into a benefit by subtracting the preference from 15 (because of the cultural houses, some people had 15th choices, which I wanted to show up as 0 benefit). This is a linear benefits assignment (as per Figure 2.6).

Iterations:

389

Number of students:

854

Average Preference:

1.46

Low preference:

5

I didn't have an authoritative source for the optimized result of this data set, but I did note that only two people got as low as their 5th choice. This seems very good, especially given the hugely unbalanced open slots in the dormitories (some dorms have 9 times as many slots as others). The average preference is very low, an encouraging sign.

3.10 Run 2

I wondered whether this positive result had more to do with the relatively unconstrained problem I had set up for myself, and so at this point I added gender balance contraints into the mix. Remember that these are in the form of hard constraints on the number of males and the number of females allowed to live in a dorm. I expected this contraint to strongly affect the results. I still used a linear benefits model, and achieved the following results:

Iterations:

252

Number of students:

854

Average Preference:

1.50

Lowest preference:

5

Fortunately, accounting for gender balance, even in such a rigid way as declaring the number of males and females to be in each dorm, did not have a big negative effect on the performance of the algorithm. The average preference remained high, and the lowest preference remained the same.
3.11 Run 3

I recalled my conversation with Tony Gray, who ran the lottery over the summer for SLP. He mentioned that he favored an approach that keeps students away from their lower choices, since most people tend to prefer a few top dorms and would rather not be assigned to a dorm below a certain ranking. Agreeing with him, and noting that perhaps the lottery would be improved if students were allowed to enter benefits directly instead of me having to guess what their preferences mean, I decided to see what I could do about those people getting their 5th choices. I reasoned that an inverse-square mapping would serve the purpose (see Figure 2.6). My results were as follows:

Iterations:
534

Number of students:
854

Average Preference:
1.48

Lowest preference:
4

I immediately noted that this method results in many more iterations, presumably because students are switching a lot among their first few choices. I also noted that the average preference improved somewhat, and the lowest preference, the metric I was trying to improve, went up.

3.12 The real assignments

I had access to the real assignments made over the summer, and attempted to analyze them in the same way as my own results. The lottery used an "annealing" process, making assignment swaps between students based on a decreasing "temperature". I took the assignment data, cut out the portions that I had ignored for my purposes, and acquired the same metrics:

Iterations:
N/A

Number of students:
856

Average Preference:
1.52

Lowest preference:
5 (9)
I was not able to compare exactly the same students between Tony's data and mine - note that there are 2 more students. This is because of students who did not enter any preferences. Nevertheless, note that the average preference is worse in the real assignments than in my Run 3, and that the lowest preference is much lower. To be fair, this is probably a fluke - one male student was assigned to Next House, a place he ranked 9th. It is possible that this was done by hand, and is not a result of the algorithm. I did note several 5th-place assignments, however, which I will take to be the correct metric.

Therefore, on all counts that I measured, the auction algorithm is superior to the annealing process by a small (but significant) margin. A difference of .04 in the average preference translates to about 40 preferences across the student body, which means that the auction algorithm assigned about 40 students to dorms that they had ranked one higher than the dorm they were assigned using the annealing algorithm. Given that some dorms accept fewer than 40 freshmen, this is a significant achievement.