Gale-Shapley

CS 3600

Submitted by: David Martin
201724804
Submitted to: A. Kolokolova

February 19^{th} 2021

Contents

1 Introduction

First, we should start with an analysis of the implementation of the algorithm. Believing that the *setup*() function is uninteresting, we will look at the data structures it creates. There are two k by n arrays, $stuPref$ and $hopPref$ representing possible preferences for students and hospitals. There are two 1 by n lists, $stuPrefList$ and $hopPrefList$, that list which hospitals use which preference lists – as described in class. There are also two n length lists, *stuAvail* and *hopAvail*, representing if the respective student or hospital has been assigned and if so, what hospital/student it has been assigned. It accomplishes this by assigning a –1 to the i^{th} element if it is unassigned and a j if it is assigned to the j^{th} element from the opposite list. The relevant parts of the algorithm are in a While-loop that checks if the sum of *hopAvail* is equal to the sum from 0 to n–1 which infers all elements have been assigned. In the While-loop there is a For-loop iterating through possible depths calling *GaleShapley*(), which has another For-loop that iterates through each hospital, checking if it has been assigned, and if action is needed checking if the student at that preference depth has been assigned or needs to be reassigned. Of note, if a student is reassigned, it resets the hospital it was previously assigned to as 'unassigned' and then moves on. This means that if the hospital whose student was reassigned was assigned at depth 0 and the algorithm is currently on depth 10, preferences 2 through 9 will go unchecked for potential matches. If this hospital is still unassigned when reaching depth n–1, the algorithm will go back to depth 0 and continue iterating through and eventually check depths 2 through 9 if necessary.

2 Where n Equals k

We can now begin the analysis of the implementation looking at instances where $n = k$ or where each hospital and student has their own preference. Below we see the table with increasing n and k on the side. The average depth represents how deep into the preference list the algorithm had to go to find a solution. If the average depth value is greater than n, it means that the algorithm had to start from the beginning again. The next column, leftovers, represents how many hospitals were unassigned when the algorithm had reached the end of the preference list. Next represents the amount of time it took the algorithm to run – the clock was started after the instance was initialized such that creating the preference list was not included in the time. Finally, there is the standard deviation for both the depth and the time. Of note, the times represent a 1000^{th} of a second and each n was run ten times and results are averages what was returned, as requested.

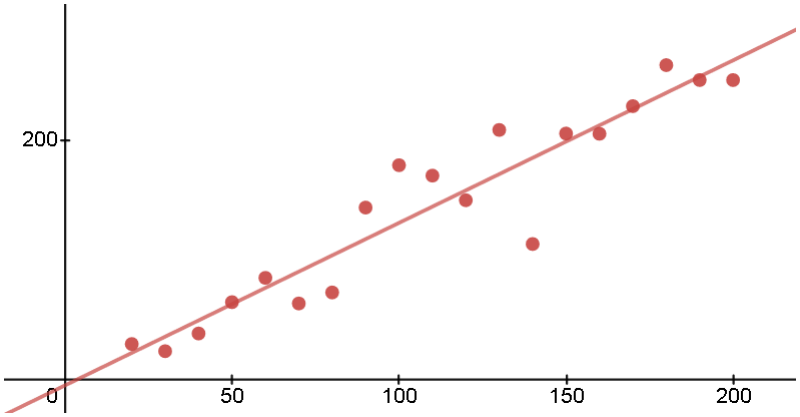

Figure 1: Required depth where $n = k$

n	Depth	Leftovers	Time [ms]	StDev Depth	StDev Time
20	29.0	0.8	0.00	15.22	0.00
30	23.0	0.8	0.28	5.16	0.89
40	37.8	1.1	0.23	20.08	0.48
50	64.0	2.0	0.5	29.66	0.50
60	84.3	2.7	1.02	48.95	2.17
70	62.9	2.9	1.85	19.72	2.63
80	72.1	3.3	0.48	41.56	0.66
90	143.1	4.5	2.36	70.51	2.78
100	178.5	5.6	0.90	72.64	1.77
110	169.8	6.9	2.96	57.03	3.40
120	149.2	7.7	3.83	72.15	1.95
130	208.0	8.9	5.32	139.46	4.19
140	112.6	9.2	1.57	71.30	3.06
150	205.0	10.1	4.84	70.70	3.40
160	204.8	10.7	3.88	96.22	4.43
170	227.9	11.6	5.69	147.13	5.52
180	262.2	12.7	9.50	103.74	3.30
190	249.7	13.3	9.01	98.05	2.41
200	249.7	13.9	8.57	92.09	6.80

As expected, the average depth increases as n increases. This follows from speculation because the chance of one of the later hospitals (in a position closer to n) would want a student that has already been requested is high. From this table we can find a function that maps n to required depth and another to the required time. To aid the software we will also give it the value of $n = 1000$ where the average depth equals 2008. Anticipating that the algorithm will be a second-degree polynomial, we will still test polynomials of degrees one, three, and four as well as an unknown power, and an exponential. We can test $n = 10000$ on each of the returned results to determine their correctness. Red represents the required depth, while blue represents the required time.

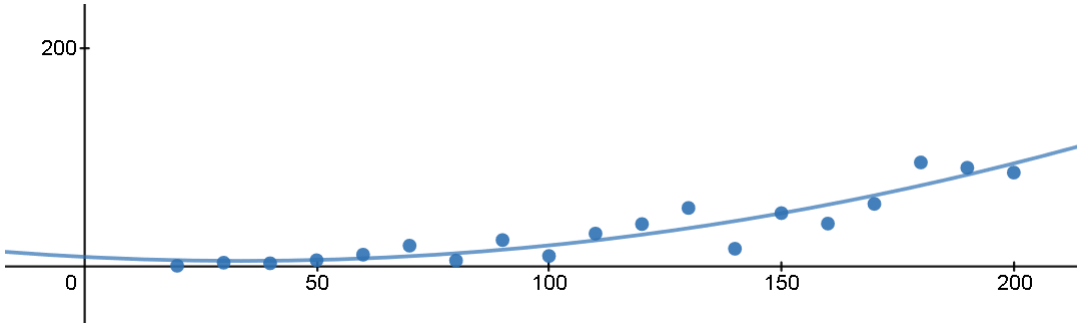

Figure 2: Required time where $n = k$

We can thankfully say that the exponential was very wrong. When $n = 10000$, the average depth was only 17394.6 with an average time of 27 seconds (note that all other times are 1000th of a second). We see that most of the equations are very wrong on their prediction for depth except for the linear function which was $\tilde{1}7\%$ greater than the actual. If this holds true it infers that the time and required depth are not actually proportional to each other, but it is rather a fluke of our small sample size. We can continue this exercise of finding an equation of best fit for time.

We see that most are extremely incorrect with the quartic even being negative. As expected, the second-degree polynomial has the truest value only $\tilde{1}7\%$ off from the actual value of 275846.40 (27.6 seconds). Adding the output from the $n = k = 10000$ case creates a functions $d = 1.74528n$–33.87 and $t = 0.00272728n^2 + 0.316034n$–41.34 for depth and time respectively.

3 Constant k

We can expand our analysis on the Gale-Shapley algorithm by having a constant k with a changing n. We will continue to use average of ten times for each row of the following table. We see that the table below matches its counterpart above. We can take a closer look at each of the values and we see that only seven of the 19 times from our constant k were faster than when $n = k$ but six of the 19 did not require going as deep. From this we can see that it is probable that having many hospitals and students with similar preferences can slow down the pairing process. This can be easily explained that there were a lot of conflicts and the algorithm often had to reassign students.

n	Depth	Leftovers	Time [ms]	StDev Depth	StDev Time
20	17.4	0.3	0.00	7.46	0.00
30	30.6	0.6	0.00	10.23	0.00
40	55.5	1.3	0.00	26.30	0.00
50	71.0	2.1	1.60	51.21	3.08
60	61.8	2.4	0.21	27.31	0.44
70	62.9	2.6	1.75	19.85	2.93
80	121.9	3.4	1.89	57.85	3.99
90	96.9	3.9	2.11	39.33	3.79
100	138.3	4.7	2.71	40.95	3.89
110	100.6	4.9	2.99	51.49	4.43
120	151.9	5.3	4.79	52.82	4.56
130	210.6	6.3	3.75	123.07	4.29
140	208.0	7.2	3.46	115.59	4.22
150	148.1	7.5	5.07	59.92	4.48
160	207.4	8.1	5.10	95.25	4.25
170	249.8	9.0	5.56	112.32	4.93
180	298.6	10	12.51	272.91	6.08
190	254.2	10.7	9.54	148.49	3.64
200	274.9	11.5	11.09	130.45	3.14
10000	16977.4	0.9	314064.31	6894.34	13786.74

4 Variable k with a Constant n

We can continue our analysis and look at the depth and time for when we have a constant n of 1000 and a variable k from 10 to 200 of increments of 10. The below table shows those results. Interestingly, we see that the time drastically decreases as does the required depth as k increases. It is reasonable to believe that this decrease is caused by the decrease in conflictions. When there are 1000 hospitals and only 10 preference lists, we can reasonably assume that there would be a lot of requesting already-assigned-students meaning their preference lists need to be checked and means although we have checked two hospitals preferences at that depth, only one has been assigned.

k	Depth	Time [ms]
1	999.0	9279.38
10	836.8	585.58
20	778.4	367.66
30	683.5	331.61
40	517.0	301.66
50	432.7	283.60
60	519.2	288.79
70	587.0	261.72
80	541.8	248.37
90	476.3	245.66
100	606.9	262.10
110	667.4	271.01
120	551.8	239.60
130	628.2	269.50
140	489.0	286.50
150	678.8	253.95
160	504.6	284.04
170	357.2	304.78
180	516.9	207.04
190	381.5	225.26
200	615.7	220.11
1000	587.9	214.83

Looking at the case of $n = k = 1000$ we see that the time and depth are very similar to that of $k = 200$ implying that any decreases in time or required depth plateau around a fifth of the n. We can confirm our suspicion that the increase in time and depth are caused by the increase in conflicts by looking at $k = 1$. We see that the depth is 999 meaning that every preference depth was considered, and we can determine that only one hospital was assigned at each depth, but every unassigned hospital would have had to also made an offer to the student to consider. We can again use Desmos to create an equation of best fit. Seeing that this plot is clearly an exponential, we will start by trying to fit an equation of the form $y = a(b^k) + c$ and we get $t = 130454(0.690786^k) + 2677$

Similarly, if we try to create a function mapping k to the required depth, we get a similar equation in the form of the function mapping k to required time, where $t = a(b^k) + c$ where values equal $t = 510.94(0.947^k) + 534$. The values are do not appear to follow a defined path but rather are between bounds so the function, while closer than others, is not very accurate as seen in the diagram below.

5 Other Analysis

5.1 Implementation Alterations

When we look at one of the original figures, of $n = k$, we can see that the left over values (hospitals that have gone unassigned even though the algorithm has reached the final depth)

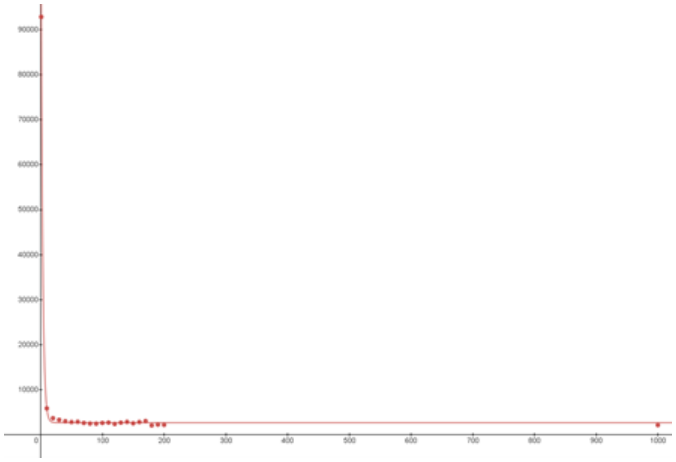

Figure 3: Required time with variable k and constant n

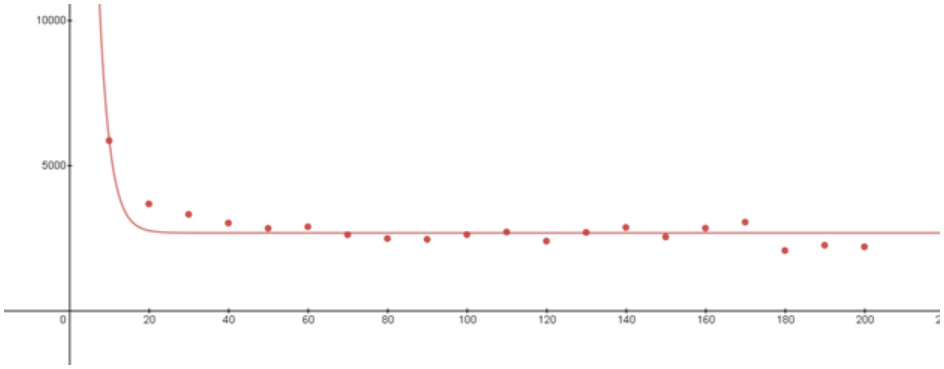

Figure 4: Required time with variable k and constant n

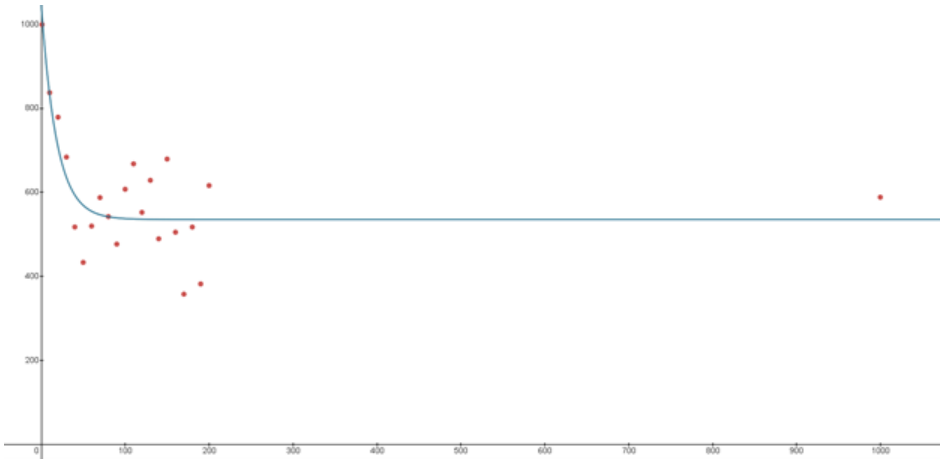

Figure 5: Required depth with $n = 10000$

are usually very small, but the depth reached on the second pass can be very extreme – reaching depth 7394 on average for $n = 10000$. We can postulate that this is in part due to hospitals having their assigned student, reassigned; so, depths from when the student was assigned to when they were reassigned go unchecked until the next pass. But what if we checked these values? In the following table, shows results from an altered algorithm where the algorithm checks the hospitals preference from when the student was assigned to when they were unassigned. Meaning if hospital H is assigned their first choice, but when the algorithm is checking preference depth 5, reassigns their student, it will now check H's second preference, all the way until it reaches the current depth 5.

n	Depth	Time [ms]	% of Preference List Used
20	11.3	0.00	56.50
30	16	0.00	53.33
40	14.2	0.00	35.50
50	33.3	0.94	66.60
60	30.4	2.56	50.67
70	38.6	1.40	55.14
80	51	1.19	63.75
90	49.1	1.00	54.56
100	59.1	1.88	59.10
110	41.7	1.70	37.91
120	59.5	2.19	49.58
130	71.9	2.41	55.31
140	66.5	3.32	47.50
150	78.6	4.54	52.40
160	80.4	5.82	50.25
170	99.9	5.85	58.76
180	107.1	6.43	59.50
190	103.1	5.30	54.26
200	102	8.11	51.00
1000	526.6	216.18	52.66
5000	1984	5025.93	39.68
10000	3973.8	19473.59	39.74

Upon comparing the values, we can clearly see that with the adjustment, that the algorithm went less deep into the hospitals preferences but was also faster than its counterpart. Part of the speed increase can probably be explained away by the algorithm not being in O(1) for finding unassigned hospitals because it iterates through the total list of hospitals checking if each is unavailable and this alteration immediately passes it an unassigned hospital. The depth though, we see as preferable. Especially to hospitals who are guaranteed a student from earlier in their list. The least favourable matching occurred in $n = 50$ where a hospital was assigned a student that was 2/3 of the way down their list in position 34. The most favourable matching occurred in $n = 40$ where all hospitals were assigned students from the beginning third of their preference list.

Finding equations to match the new output, we find another linear function for the depth of $d = 1.721n^{0.8293}$–23.59 and a second-degree polynomial $t = 0.1927n^2 + 0.1927n$–21.34.

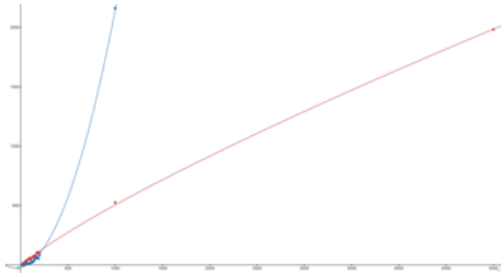

Figure 6: Required depth and time of altered algorithm

These functions were produced without the $n = 10000$ instance. When we evaluate both at $n = 10000$, we get the values 3548 and 199176 which are about 12% and about 2% off from the actual values. In the graphs below, the blue represents the time while the red represents the required depth.

5.2 Condensed Values

Moving on, we can consider the case where a group of hospitals or students is seen as being undesirable. In this situation, hospitals and students numbered greater than $\frac{n}{2}$ will always be in the later half of the preference list. This means that the first half of hospitals will always be chosen by students to accept their offers. We will continue to use the adjusted implementation where hospitals with reassigned students are immediately assigned a new student.

n	Depth	Time [ms]	% of Preference List Used
20	16.5	0.00	82.50
30	24.3	0.00	81.00
40	31.3	0.65	78.25
50	37.5	0.00	75.00
60	46.5	1.57	77.50
70	56.3	3.13	80.43
80	61	7.80	76.25
90	67.1	4.69	74.56
100	74.3	7.81	74.30
110	80.6	6.25	73.27
120	93.9	7.32	78.25
130	95.6	13.15	73.54
140	106.9	19.13	76.36
150	113.6	13.97	75.73
160	122.9	16.66	76.81
170	140.2	21.34	82.47
180	130.5	22.57	72.50
190	153.2	23.59	80.63
200	141.8	27.87	70.90
1000	746.3	1856.35	74.63
5000	3687.8	165744.31	73.76
10000	7402.2	1143731.41	74.02

As compared to the previous table, where all assignments from the first two halves of their preference lists – now the iteration that went the least far into their preference list was where $n = k = 200$ which required 70% of the list. Further, every iteration was slower than its counterpart from the first section with the other $n = k$ table. We can confidently say that having these generally-agreed-upon unpreferable students and hospitals will slow down the assignment process. We will again use Desmos to graph the results from this experiment. The red will represent depth while the blue will represent time.

We can again see that the depth is linear while time is a polynomial degree 2.78. The exact equation given by Desmos is $t = 8.15(n^{2.787}) * 10^{-6} + 3.17$ and for depth is $d = 0.73n + 2.44$. Algebraic analysis shows that the required depth for this restriction on the preference lists will mean the algorithm will always be required to go deeper into the preference list and will always take longer.

5.3 Reversed Lists

We have looked at instances where many hospitals shared lists and where hospitals agreed that some were less preferable than others – but what about instances where hospitals and students are directly opposed to each other? We will create a scenario where if a preference list exists, the reverse of the list also exists. For example, if Hospital A has the list [2,3,1,4,5], hospital B will have the list [5,4,1,3,2].

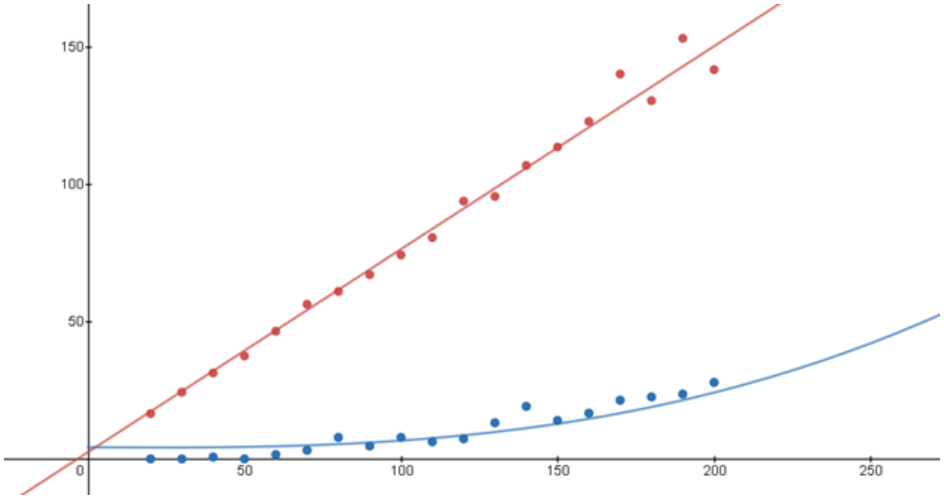

Figure 7: Required depth and time of condenced values

n	Depth	Time [ms]
20	10.6	0.10
30	18.3	0.20
40	20.4	0.60
50	31.5	0.69
60	36.1	0.89
70	39.5	1.06
80	38.7	1.30
90	52.7	1.50
100	57.1	1.71
110	43.4	2.30
120	48.7	3.19
130	64.6	2.59
140	57.1	3.00
150	64.2	2.79
160	74.3	3.97
170	92.3	6.54
180	94.0	5.68
190	71.6	6.12
200	120.1	5.79
1000	539.9	233.34
5000	1876.6	4947.94
10000	3873.3	20126.57

Comparing these results from the table at the beginning of the Other Analysis section, at first glance there was no increase or decrease in speed. When looking at specific n's half were faster than the original, half were slower. When we look at the required depth of this instance, there was also little change. 14 of the 22 did not require going as deep while

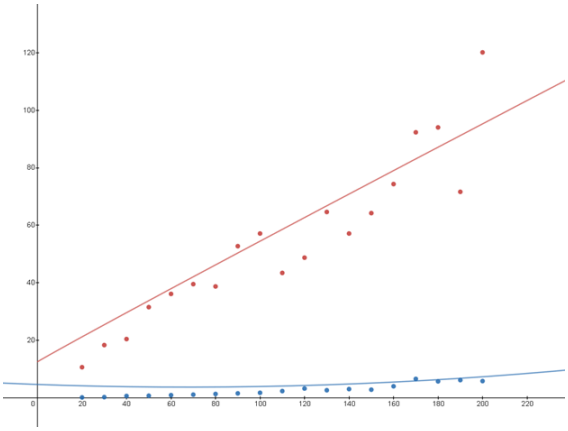

Figure 8: Required depth and time with reversed lists

the other 8 went deeper. But, when we look at the equations created by Desmos, we can determine that the required time, while similar at the beginning, will diverge. The equation returned by Desmos for time is $t = 0.0002n^2 - 0.027n + 4.6$ and depth is $d = 0.458n^{0.98} + 12.5$. One reason for this might be because it reduces the number of conflicts. It forces a scenario where every hospital and student has a counterpart that it does not compete with.

6 Conclusion

From our analysis, we can see what effects the run time of the Gale-Shapley algorithm. We see that a low k – few discrepancies in preference lists – can drastically increase the run time. One method to address this might be to compile a list of hospitals that prefer a student at a given depth, then checking that students preference list and assigning the student their preferred hospital who is about to make an offer. Further analysis could be done on a situation where a student decides they no longer want to be apart of the program and backs out. Is there an efficient method to check if the solution is still viable? Or must the algorithm restart?