Don E Merson

dmerson@email.arizona.edu

Abstract

While many matching algorithms in academic situations optimize for many different ends, there hasn’t been any research into optimizing the final result for fairness. This paper examines creating seven algorithms based on common matching scenarios to allow a user to compare each possibility and rate the final results on ideas for fairness.

Optimizing for fairness in Scholarship Awarding System

# Introduction

Matching algorithms are an important part of graph theory that can be applied to real world problems. Examples of algorithms such as the Gale–Shapley algorithm have been used to solve real world problems like the matchings of Residents to Hospitals and Applicants to Schools (The Royal Swedish Academy of Sciences 2002). One of the most important matching problems is the scholarship awarding process. Matching students to scholarships is an important component of many schools plans to help their students and institutions succeed.(Belloni et al. 2018) It is in fact a type of optimization problem where schools try to optimize the process towards other goals for the institution such as enrollment averages(Belloni et al. 2018).

The scholarship awarding process is a process where a committee matches awards for many scholarships to many applicants. However, different committees can desire different end goals but there are commonly wanted outcomes such as the student ranked are most deserving getting the most monetary awards. However, due to some quirks in the scholarship awarding process such as differing scholarship award amounts and not all student being eligible for all scholarships, some basic logical oddities can arise when one looks at the final results after awarding multiple scholarships. These include having higher ranked student receiving less than lower ranked students or no awards at all. Thus, these results don’t get optimize for perceived fairness to the quality of the students.

To combat this problem, I explore seven scholarship matching algorithms and see how they compare when run in tandem to determine if they can fix these common pitfalls. These algorithms are based on common practices for awarding committees such as merit based, split based, minimum awards, and maximum total awarding for students. I implement these seven algorithms in a MS-SQL Database with a normalized version based on the work of Codd (Codd 1990) and a denormalized version which would allow other external scholarships easy input into the working of the algorithms. Once these seven algorithms are run, I can examine the results to look for logical consistencies that are being adhered to by using certain algorithms. There are three logical consistencies which I call the rational assumptions (RA) that be checked for each run of the seven algorithms. The end results is that a simple query can be run that can examine and decide which algorithm is best for the given data.

By implementing these algorithms with the SQL database language, it allows other languages with ODBC Drivers or native MS-SQL Drivers to hook into this work allowing to be easily expanded and commercialized in the future.

# Related Work

briefly describe matching algoirthms used in this work. How are they similar or different from yours?

Matching and optimization algorithms have been used as a subject of research often. One of the most common cited is the Gale-Shapley algorithm which resolves the stable marriage problem (Teo, Sethuraman, and Tan 2018; Shapley and Roth 2012). This work has considered important enough to help the Shapley and Roth to receive a Nobel Peace Prize in Economics.(Nobel Price Committee 2017) However, this work has optimized information based on economic related goals such as the stability of matches (Biró 2018).

Some of the work around matching algorithms looks into using it for school admissions (Biró 2018; Gale, D and Shapley 1960; Teo, Sethuraman, and Tan 2018) The original paper by Gale and Shapley looked into college admissions and showed the algorithm would be a stable solution (Gale, D and Shapley 1960).Brio looks at matching in Hungary which has a centralized system for matching.(Biró 2018) The goal of this paper was to verify that the matchings are stable (Biró 2018). Teo et al look into the school assignment process in Singapore (Teo, Sethuraman, and Tan 2018). They find that the current process is not optimal and how the algorithm can better match the students (Teo, Sethuraman, and Tan 2018) which was the original contention of the 1960 paper by Gale and Shapley.

However, Alvin Roth has noted that “some important properties of the problem of the marriage problem, do not, as previously believed, carry over to the college admission problem.”(Roth and Sotomayor 1989) His work improving upon the solution was why he shared the award the Nobel Prize also. (The Royal Swedish Academy of Sciences 2002)

Matching algorithms can also lead to considerations of resource allocations. In the Stable Marriage Problem, the final goal is not any type of end allocation only that the final result will be stable. Belloni et al look at the consequences of scholarship allocation on the college admissions process. Their paper looks at the process of awarding a scholarship as a price discrimination problem (Belloni et al. 2018) They look beyond the matching problem and look at the distribution of the group as a whole as part of the problem needed to be understood. What is similar to this paper’s research agenda is that the schools looked at the final averages of the student body to be part of the judgement of how well the optimization has performed. (Belloni et al. 2018)

However, the most interesting and relevant to this paper is Rachmawati’s 2017 paper on matching student to scholarships at the University of Oklahoma. (Rachmawati 2017). Rachmawati “focuses on how to optimize the matching of scholarships and students, taking into consideration the requirements of the scholarship and the credentials of students who are applying for the scholarship (Rachmawati 2017).” However, there are many services that automatically match students to scholarship with algorithms. With this in mind, I consider matching due to requirements of the scholarship a problem with many solutions within many commercial websites and one that I solved with Scholarship Universe at the University of Arizona. Rachmawati’s work is interesting but only takes into consideration the matching of a particular scholarship from a particular donor and matching this to the pool of students. It doesn’t look at the fairness of the final result.

# Problem to Be Solved

This paper concentrates on a problem after all of the students in a particular committee have matched all the scholarships. The question I am interested in whether this final distribution of funds matches the committee’s opinion of the student rankings. This current work is going to take for granted that all students are already matched with all the proper scholarships within a given scholarship system. This work also doesn’t assume that the values for a requirement to be eligible for a scholarship should have any weight because there are other factors in scholarship processes such as application essays, resumes, etc. that need to be ranked by the committee. It is also important to note that even objective measures such as GPA can greatly influenced by factors such as difficulty of the major and current academic level. For this reason, this work will assume that a committee has ranked a group of student eligible for scholarships with whatever criteria they desire whether it is academic merit and/or financial need.

The Scholarship Awarding process can be describe in formal terms as a matching algorithm of a bipartite graph of Scholarships S and Applicants A. After the matching has been completed there is a result set of Applicants and the amounts they have been awarded.

Scholarships are nodes which begin with S and applicants are nodes that being with A. The numbers are a ranking for the scholarship or application. For example, S2 would be the 2nd highest ranked scholarship and A2 would be the second highest ranked student. Student rankings are decided by the committee and not part of the matching algorithm. The committee may use any means to rank them including academic merit, financial need, or brilliance of an essay. Scholarship rankings are determined by higher award amounts. If the case of ties, committees determine the higher ranking by other factors. It is important that there are no ties between scholarships and applicants.



Figure 1Scholarships and Applicants

An edge(line) between a scholarship and applicant which means that the student was qualified for a given scholarship. Not all students are qualified for all scholarships.



Figure 2 Edges show qualified applicants

So in the previous image, S1 has applicant A1, A2, and A4- S2 has A1 and A3-and S3 has A1, A2, A3, and A4.

A bolded edge means that the applicant was awarded the scholarship.



Figure 3 Awarded Scholarships

In this case S1, S2, and S3 are awarded to applicant A1.

To the left of all of the scholarships is the amount of the scholarship. To the right of the student is the final amount of scholarships that the student has won after awarding.



Figure 4 A1 is awarded 2250

The previous algorithm for Figure 5 is called the Merit Only algorithm (which is discussed in detail later in the paper). Each algorithm has a set of steps to determine which applicant is awarded each scholarship.

The problem is that after algorithms have been run on a large dataset of scholarships and applicants, illogical results can result that are contrary to the intuitions of the committee. One major example is that a highly ranked student can have been awarded nothing while a lower ranked student can have many awards. Or as figure 4 demostrates it is possible that only one highly ranked student could be awarded all of the scholarship awards. The seven algorithms algorithms that are explored in this paper are run against a dataset to show that a different tweak to the algorithm can lead to a more intuitive and fairer result.

After all the awards have been determined, an query will be run against the various datasets to determine if they match up to desired outcomes The algorithms are designed to find solutions where these oddities don’t exist. We have three end states which are desired called the three rational assumptions.

The first desired assumption is to assume that a higher ranked student should have more awards than a lower ranked student should. We call that rational assumption 1 or RA1. For example, the top ranked student should end up with more awards than the 2nd ranked student,etc. (See Set 1 in the figure below)

The next rational assumption , RA2, would be that students with rankings next to each other could have an equal award amount or the higher ranked student would have a higher amount. For example, the 2nd and 3rd ranked students might have the same amount. (See Set 2 in the figure 5 below)

The last assumption, RA3, would be that every student that there would be never be a higher ranked student without any awards when a lower ranked student would have an award. (See Set 3 in the figure below)

Set 4 shows a result where none of the rational assumptions is met.



Figure 5 Examples of Rational Assumptions

Given a graph of data the goal is to determine which algorithms are attuned to one, two, or three of the rational assumptions to help the committee to avoid illogical awarding decisions if possible. It is always possible in a given dataset that no rational assumptions can be found in the data. However, it is also possible to tweak some input parameters to the algorithms to try to find a solution if the committee doesn’t succeed at first.

The end analysis will also include other fairness measures such number of unique students who have been awarded which will consider more fair the higher the number. We will also track the minimum and maximum final awards for the set as gauge for the distribution of the awards.

# Methods

## General Ideas of Seven Scholarship Matching Algorithms

There are seven algorithms based on real world preferences of real scholarship committees that will be explored. The ideas brought from real world committees are: merit based awarding, maximum awards, minimum awards, and splitting awards among applicants. After introducing each of these algorithms briefly, I will examine each algorithm in more detail.

The seven algorithms are as follows:

* **Merit only**-This algorithm just applies the scholarship’s highest ranked applicant the award. This has the issue of allowing one qualified individual to crowd out all other applicants.
* Maximum Awarding Preferred Applicant-These two algorithms apply a limit to the amount an applicant can require.
  + **Merit Only Awarding Disqualify after Exceeding Maximum**-After exceeding a maximum, the candidate is removed from the applicant pool for other scholarships.
  + **Merit Only Awarding Can’t Exceed Maximum** - If awarding would exceed a maximum amount, the applicant is considered not allowed to be awarded a scholarship. This doesn’t remove the applicant from other scholarships.
* **Maximum One Award Per Applicant** - An applicant is only allowed to be awarded a single scholarship.
* **Split with All Qualified** - The award is split among all the applicants for a scholarship. This can lead to excessively small awarding amounts for applicants.
* **Split with Minimum Qualified Applicant**- The award is split to a fixed number of applicants and the award is split among those applicants.
* **Split with all qualified applicants with minimum amount given** - The award is split to all qualified applicants when the split will not be below a certain minimum awarding amount.

Now let’s examine each algorithm in more detail with a graph and example.

## 1.Merit Only Awarded

Highest ranked gives the scholarship to the highest ranked applicant without any other considerations. 

Figure 6 Merit Only

In this example, A1 gets awarded each scholarship. Note that this is valid for rational assumption 1 (RA1).

## 2.Merit Only Awarding Disqualify After Exceeding Maximum

This method awards via merit but after a candidate reaches a maximum award, they are removed from the applicant pool for lower ranked scholarships.



In this example, A1 is not qualified for S3 because they had already earned $1750 and thus was removed as an application for S3. This allows A2 to secure the $500. This result is also valid for RA1.

## 3.Merit Only Awarding Can’t Exceed Maximum

The method awards via merit but doesn’t allow an applicant to exceed a fixed amount (In the case of this graph 1500). Therefore, if the earnings of the applicant plus the award amount exceeds this amount, they are removed from the applicant pool.



In this example, A1 doesn’t qualify for S2 because the $750 would be added to the $1000 already earned from S1 to exceed the maximum of $1500. However, S3’s award of $500 is still within the maximum so A1 is awarded S3. Note in this example, the results do not follow the rational assumptions.

## 4.Maximum One Award Per Applicant

In this method, the candidate is removed from future applicant pools once they have received any award.



In this example, A2 is not qualified for S2 which allows A3 to exceed the final award amount of A2. This example only satisfies RA3.

## 5.Split with All Qualified

In this method, the scholarship amount is split by the number of all qualified candidates. For example, a $1000 scholarship with 3 applicants would be an award for each candidate for $333.33.



In this example, we first must calculate the value for the award for a given applicant by dividing by the amount of qualified applicants. S1 is $1000 divided by S1 for $333.33 each. (Note you cannot exceed a scholarship award amount so the penny is left over) S2 is $750 divided by 2 for $375 each. S3 is $500 divided by 4 applicants for $125 each. When these are all added up, the results are only for RA3.

## 6.Split with Minimum Qualified Applicants

In this method, the award is split by a fixed set of applicants (in this case 2) and awarded to the top candidates. So the award for a $1000 scholarship with 2 applicants would be $500.



In this example, we have to calculate the award amount by divided each award amount by 2. Note that after we add up these results that it satisfies RA1.

## 7.Split with all qualified applicants with minimum amount given

In this method, the scholarship amount is split among all qualified candidates as long as the split exceed a minimum base award amount. For example, assuming a $250 minimum, a $500 scholarship with 4 candidates would be split to 2 awards of $250. However, a $1000 scholarship with only 3 candidates would be split to $333.33. If there were a forth candidate, the $1000 dollar scholarship would be split to $250. But if there were 5 candidates, the award would stay 4 awards of $250 each.



Once again we have to calculate each award first. In this example, each S1 and S2 award can be divided by the number of applicants within getting below the minimum. However, S3 has 4 applicants but can only be split 2 ways to stay over 250 minimum. Note that these results also satisfy RA1.

## Data Models

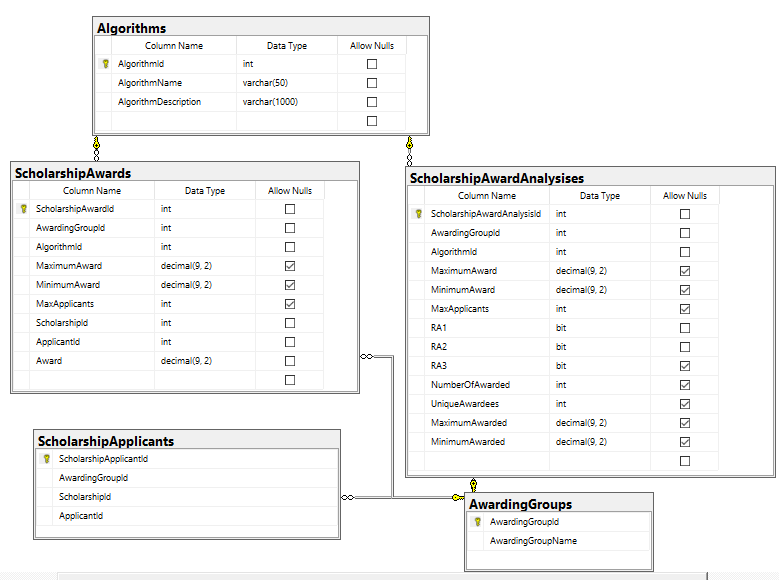
To implement these algorithms within the SQL language (MS-SQL Variant), it is required to create a data model. However, instead of only implementing the algorithm in a normalized model, I decided to create a denormalized model to allow easy import of data from a CSV import.

## Normalized

Awarding Groups is where a committee name and ID is stored. Every Awarding Group can multiple ApplicantRankings and ScholarshipApplicants. An ApplicantRanking points to an AwardingGroup and Applicant with a ranking from the committee with a surrogate key that points to this particular instance. Applicants is just a personal information of first and last name with ID. The ScholarshipApplicants table points to a AwardingGroup and Scholarship with a surrogate id. The Scholarships table hold an ID, Name and the amount of the award for the scholarship.



To store the results of running the algorithms the linking table is the ScholarshipApplicants. The ScholarshipAwards table stores the algorithm and the parameters that were used for the resulting row. It also stores the resultant award. It should noted that the algorithm can change the final award amount and one cannot just link this value from the Scholarship table through the ScholarshipApplicants table. This table links to the Algorithm table which has a name and description for each of the seven algorithms. Once all the awards have been awarded for a given algorithm and parameters, the results are then stored into the ScholarshipAwardAnalysises table. This table stores the algorithm and all the parameters, a link to the ScholarshipApplicants, and a Boolean value for RA1, RA2, and RA3. The table also stores useful information about the results such as the number of students awarded both total and unique, and the maximum and minimum total awards for any of the students in the analysis group.

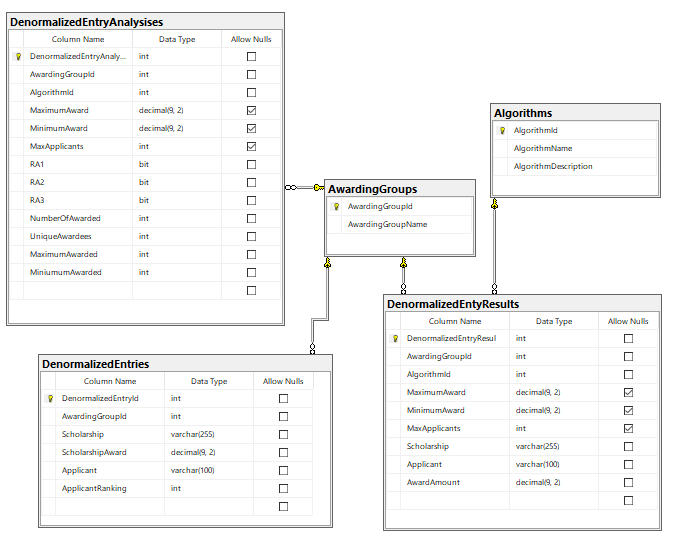


## Denormalized

Since it would be difficult for an outside system to important a normalized model due to the many surrogate keys that would need to be linked, I created a denormalized less strongly typed version that could allow easy import for analysis.

The only data requirement before import would be to grab a unique ID from the awarding groups. After this the data for Scholarship and Applicant can be a generic text fields. The ScholarshipAward can be a decimal and the Applicant Ranking must be an integer. The process of important has the ability to preprocess the imported data to verify that each user has a unique ID (which is required for the algorithms to run correctly). If scholarships have the same awarding amount, it would be up to the user importing the data to rank the more important ones by placing them first in the imported data.

Once the data is imported into the DenormalizedEntries table the same process for results for the algorithms and analysis is available for these new entries.



To compare these algorithms, I took data from the latest award year of the University of Arizona’s Scholarship Universe (SU) for the various awarding committees and compared them to the results from the algorithms. The SU data did not have a ranking of the students in the full awarding committee but I was able to determine the number of applicants, number of awards, maximum award and minimum award for the various groups.

To use the seven algorithms I was required to create a method to rank each applicant in the awarding group. To do this I used an average of the GPA/ 4 and Required Contribution ranked on their FAFSA/ Total Tuition. Any ties I just used the student ID to create a unique ranked set for each awarding group. This is an exact ranking but is close enough to give a fair assessment though in real life the awarding committees would have to rank more abstract items such as biographies, resumes, and essays to derive the final ranking.

To maintain privacy for all parties involved each student, scholarship, and awarding committee are given an arbitrary number as an ID. This will prevent any possibility to derive the identity of any student or committee which allows me to share this data in my public data repository. The SQL for these queries is also included in the repository.

Once these rankings are created, I need to determine which values should be tested with the various awarding groups. I determined that two times the maximum award for the highest paying scholarship will be the maximum award parameter. For the minimum amount I took one half of the minimum of any scholarship. For the maximum applicants parameter I took the maximum number of applicants awarding for any scholarship within the awarding group or 2 whichever is greater. With these values set I did an analysis of every awarding group and compared it to the actual awards for the group.

I expected to see certain patterns within the data to appear. I will rank as more any results with the following sort order: RA1 true, RA2 true, RA3 true, number of unique applicants, number of awards, maximum amount to applicant, and minimum amount to applicant. So for example, an result with RA1 as true with 10 unique applicants is more fair than a result with RA1 with only 8 unique applicants.

## Design of Program

The implantation of the seven algorithms within normalized and denormalized data models within a SQL model allows for the most flexible method going forward. Since most languages has some version of ODBC or a native MS-SQL Client, a framework can built that leverages the work already done within this repository. I have included worksheets within the repository for R and Python that connect to a SQL Server as demonstration of this fact. The database is using a modular design to allow for reuse and further development. All functionality from entering to anlaysing the data are possible with strored procedure to manipulate data. Since there are separate denormalized and deonormalized table, there are separate stored procedures for similar functions such as CreateAnalysis and CreateDenormalized Analysis.

All the algorithms are designed as standalone modules which are called by stored procedure that calls each algorithm in turn. The design allows new algorithms to be added with the architecture of RunAllAlgorithms and RunAllDenormalizedAlgorithms. Any additional algorithms just need to be added to the database and be called by one or both of the “Runall” stored procedures. The design of all the top level stored procedures such as RunAllAlgorithms, CreateAnalysis, GetAnalysis, GetScholarshipAwards, and RunAllAlgorithms are designed to be run and if the data is in the denormalized tables, the procedure will call the related denormalized stored procedure eliminating the chance the user will call the wrong stored procedure.

The whole process can be run easily from the stored procedure GetAnalysis which has a parameter that allows one to run the analysis first or just pull the analysis. You just input the awarding group id and the regular input parameters and the stored procedure will determine whether to use the denormalized or normalized table by reading the awarding group id.

After determining which algorithm is the preferable algorithm, there are stored procedures that allow you to pull the results for that algorithm with the same parameters called GetScholarshipAwards and GetDenormalizedScholarshipAwards respectively.

## Hypothesis 1 (H1)

I would expect that different awarding committees will have different algorithms be ranked as the most fair. If only a certain algorithm comes up in all the committees then it would be realized that this one algorithm is the most equitable algorithm which would provide proof that the seven algorithms are not needed to access a fair distribution.

## Hypothesis 2 (H2)

I would expect that all of the algorithms, except the possibility of the algorithm 1, will have more unique applicants than the original awarding group data.

# Results

Results are a set of implemented algoirthms. It would be better to run an experiment using existing data to compare the differences (fairness) in the outcome of the algorithms.

I feel the research has not started yet -- you have prepared research instruments -- the algorithms. In practice, which algorithms work better and why? If you already know which algorithms work the best, then why develop 7 algorithms

To determine whether this set of algorithms would work in real life situations I pulled data from Scholarship Universe, a full featured scholarship application at the University of Arizona. I picked a year and pulled data on awarding groups. Awarding Groups in this case could be colleges, foundations, donor organizations, or campus groups. There were 21 different awarding groups with applications in SU in that year. I anonymized their names by using the ROW\_NUMBER() function on a surrogate ID key for the awarding groups. This allows me to include the data in the repository for public consumption.

There were 21 groups that had at least scholarship applicant. However, only 13 of these groups actually used the awarding feature and thus had no information on awarding for that award group. The number of total applications ranged from 31 to 46,785. The number of distinct applicants ranged from 31 to 3437. The distinct people awarded within an awarding group ranged from 1 to 663. The max award per person was in a range of 1000 to 12548 while the minimum ranged from 100 to 2750.

However to pull this data into the database, there needed to be ranking set for each of these awarding groups. I used a average of GPA/4 + Federal Financial Need/Federal Cost of Attendance. When data was not available for each side, I treated blanks as .5. However, even with these measures there were reasons that all of the data could not be important. Once reason certain data was not imported was because a total amount for the scholarship was not entered into the system and my query removed these scholarship. Also, some applicants did not have data for each of these features (not 0) and were removed from the applicant set. Each awarding group had a lower number of rankable students who were pulled into a spreadsheet to allow an import into the database.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Awarding Group Id | Total Applications | Distinct Applicants For Group | Distinct  Awards | Distinct Awardees | Max Award | Min  Award | Rank-able |
| 1 | 6995 | 505 | 275 | 211 | 10000 | 226 | 4121 |
| 3 | 46785 | 3437 | 421 | 397 | 10860 | 157 | 23065 |
| 4 | 322 | 264 | 26 | 25 | 2000 | 500 | 57 |
| 6 | 900 | 355 | 90 | 61 | 10000 | 480 | 538 |
| 7 | 24983 | 824 | 663 | 482 | 17000 | 100 | 15779 |
| 9 | 4547 | 350 | 306 | 226 | 6005.2 | 600 | 3505 |
| 11 | 112 | 29 | 15 | 13 | 5000 | 500 | 110 |
| 12 | 1114 | 127 | 79 | 56 | 8800 | 400 | 991 |
| 13 | 534 | 121 | 63 | 52 | 2333.33 | 415.54 | 440 |
| 14 | 144 | 61 | 1 | 1 | 3750 | 3750 | 61 |
| 16 | 4886 | 376 | 460 | 361 | 12548 | 2548 | 4226 |
| 18 | 1503 | 1064 | 92 | 91 | 5000 | 333 | 302 |
| 21 | 31 | 31 | 1 | 1 | 1000 | 1000 | 27 |

The data for the students was also anonymized by using the ROW\_NUMBER() method on the student id (an arbitrary number) which gave each student a name of a number, their order in the awarding group, which keeps every student unknowable. Then, using the previously mentioned formula their ranking was determined within their group which again gave the students a number. Due to data becoming missing in the import process, there were skips in ranking (which is not allowed) which were fixed later in the import process.

I used a R Script to import each spreadsheet into the database which in turn fixed the applicant ranking numbering issue. As I imported each spreadsheet, the awarding group was analyzed to determine the fairness of the data for each algorithm. I used the parameters of 12240 for maximum amount (which is a year’s tuition), 250 minimum award, and 2 applicants as my parameters on the import.

After running all of the imports with these parameters, H1 was confirmed as there was a different algorithm. H2 also was true with most of the results from algorithms as seen below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Group Id | Original Distinct Awards | O  Max | O  Min | Top  Alg | RA1 | RA2 | RA3 | # total | # unique | Max | Min | More Awards(H2) |
| 1 | 211 | 10000 | 226 | 5 | 0 | 0 | 1 | 4121 | 364 | 28632.96 | 4 | 153 |
| 3 | 397 | 10860 | 157 | 5 | 0 | 0 | 0 | 23065 | 1773 | 105269.9 | 1.22 | 1376 |
| 4 | 25 | 2000 | 500 | 7 | 0 | 0 | 1 | 53 | 42 | 1771.74 | 250 | 17 |
| 6 | 61 | 10000 | 480 | 5 | 0 | 0 | 0 | 538 | 234 | 8153.92 | 13.28 | 173 |
| 7 | 482 | 17000 | 100 | 5 | 0 | 0 | 1 | 15779 | 556 | 28783.39 | 12.54 | 74 |
| 9 | 226 | 6005.2 | 600 | 5 | 0 | 0 | 0 | 3505 | 261 | 6671.32 | 254.81 | 35 |
| 10 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 30000 | 30000 | 1 |
| 11 | 13 | 5000 | 500 | 5 | 0 | 0 | 1 | 110 | 28 | 1899.05 | 105.26 | 15 |
| 12 | 56 | 8800 | 400 | 5 | 0 | 0 | 1 | 991 | 112 | 4204.96 | 16.13 | 56 |
| 13 | 52 | 2333.33 | 415.54 | 5 | 0 | 0 | 1 | 440 | 97 | 3609.75 | 116.67 | 45 |
| 14 | 1 | 3750 | 3750 | 1 | 0 | 1 | 0 | 9 | 4 | 31200 | 5000 | 3 |
| 15 | 0 | 0 | 0 | 5 | 0 | 0 | 1 | 9828 | 211 | 10827.37 | 1.32 | 211 |
| 16 | 361 | 12548 | 2548 | 5 | 0 | 0 | 1 | 4226 | 305 | 6464.05 | 158.76 | -56 |
| 18 | 91 | 5000 | 333 | 5 | 0 | 0 | 0 | 302 | 193 | 10428.86 | 62.5 | 102 |
| 19 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 500 | 500 | 1 |
| 20 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | 3 | 2 | 20000 | 15000 | 2 |
| 21 | 1 | 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 | 1000 | 0 |

However, as noted above awarding group 1 finishes with a minimum award of 4 and awarding group 3 finishes with 1.22 and this is too little money to actually be fruitful. So if we take the next best algorithm for awarding group 1 which is algorithm 7 we finish with a minimum award of 226 with 327 unique users which is still 101 more unique users than the original method used by the awarding group. If we take the second place finisher for awarding group 3, we have a minimum award of 250 with 714 unique award winners which is far more than the original 397.

# Discussion

The research project has shown that with real life data that the algorithms can find ways to award more students while still staying fair to all parties involved. The strictness of the RA standards was met by 12 of the 17 awarding groups that were tested. However, to meet this strict standard sometimes the awards were made into two small of sums which is reason for the user to look at the results for each algorithm for the analysis.

The most common winner was algorithm 5 which splits with all qualified. In cases where the number of applicants is low this can be a result that gives a decent amount to every person. When the set is small, Algorithm 1, merit only, can easily work but having a maximum was shown to be an effective method which is why algorithm 2, merit only disqualify afterwards meeting threshold, wins awarding group 20.

It should be noted that algorithm 7 split with minimum qualified was highly ranked consistenly finishing behind algorithm 5 in almost every instance. Since this allows the user to tweak the minimum amount given, this algorithm creates a powerful result while keeping the scholarship per student at a reasonable rate.

Algotihm 6, split with minimum qualified applicants, never finished higher than 3rd in any of our awarding groups. However, the parameter we used was only 2 and if a higher number were chosen it is likely that it would perform more like a less excessive algorithm 5. It could arguably be taken out of the possible solutions because even when I tweak the maximum applicants parameter, the algorithm never performs better than the rest.

Algorithm 4, Maximum One Award Application, also never finishes higher than 3rd and usually at the bottom. It never once reaches a RA standard. For this reason, it seems that algorithm 4 should be considered sub par and could be taken out of running for the seven algorithms.

# Conclusion

In conclusion, it would seem the next bit of development would be to remove algorithms 6 and 4 from the mix because they are computing intensive but are unlikely to be the final solution.

There are some improvements that can made to this grouping of algorithms in the future. The most obvious is the implementing of the algorithms within a procedural, object oriented, and functional languages. Since the algorithms are already defined, this shouldn’t pose any issues. The languages that are chosen for further development at this time are respectively R, C#, Python, and JavaScript as the next stops in the implementation of these algorithms. All four languages have native MS-SQL Drivers that can allow the first step to be a database driven version. I have already started by giving examples in R and Python. In fact, I ran the experiment using R to call the stored procedures.

Going away from declarative languages and toward procedure languages there is an opportunity to become more flexible with the rankings of the scholarships. Most real world scholarship systems have many scholarships with the same award amounts. This allows the implementation of different orders for the scholarships amount. For example, if there are 3 scholarships with $1000 awards labeled A, B, and C. The algorithm can first create scholarship lists with ordering such as:

1. A, B,C
2. B,A,C
3. C,A,B
4. A,C,B
5. B,C,A
6. C,B,A

However, this will quickly expand the scholarships that must be analyzed for the algorithm quickly. As the previous example shows, instead of one run through the list, the algorithms would need to be run 6 times. This would also expand the listings the end user would have to pick to determine the best choice. In the previous example, if we add three more scholarships with the same amount of $500 dollars, our number of scholarship possibilities expands to 36. It is not clear at this time if this added complexity would actually help out the end user of the product or make the decision as complex as the current situation without the algorithm help.

Thirdly, one could add to the algorithms including having mixtures of the various current algorithms. Once again, this hybrid or new ideas would have to be weighed in balance to the end result of making easy for the end user to identify the most fair and logical choice.

# Appendix-Algorithms

To implement this programmatic intention, I created a MS-SQL Database and Stored Procedures to carry out the logic for the methods. There are two versions of the algorithms to match the normalized and denormalized versions of the data schema. I used the Red Gate SQL Source Control tool to save this schema to a GIT Repository which includes documentation and worksheets which allow the user to step through the code instead of running the stored procedures. I also used the Red Gate Compare SQL and Data Compare tools to save copies of the basic data for testing and the changes in the schema as I developed and made changes to the database. This repository is located at <https://github.com/dmerson/ArizonaMastersCapstone>. The repository contains creation sql scripts to be able to create the database and 2 basic awarding committees in the read me file. The user can recreate the database by following the directions of the read me on the site.

Each algorithm has a stored procedure that corresponds to its ID in the algorithm table. For example, the row in the algorithm table with the ID of 1 corresponds to RunAlgorithm1 stored procedure which run the Merit Only algorithm. There is a denormalized version of this if the data exists in the DenormalizedEnties table called RunDenormalizedAlgorithm1. There is a stored procedure that can run each of the seven stored procedures at once taking in the parameters of :@awardgroup INT, @MaximumAward DECIMAL(9, 2), @MinimumAward DECIMAL(9, 2), and @MaxApplicants INT. There is one stored procedure to run the normalized versus the denormalized versions of the algorithm.

I am going to discuss the base algorithms and how they are implemented in code. Certain algorithms don’t need input parameter but they are required when comparing the finalized result analysis. For example, algorithm 1-Merit Only doesn’t require any parameters but when later comparing it to other possibilities within the seven algorithms it is required for the comparison analysis query. For these reasons, I am including them as I will include them in the detailed analysis of the algorithms below. To simplify this I will use the phrase “Get Input Parameters” with the explanation of the algorithms. This group of parameters equates to: :@awardgroup INT, @MaximumAward DECIMAL(9, 2), @MinimumAward DECIMAL(9, 2), and @MaxApplicants INT. Each algorithm ends with an input into an analysis table which has calculated RA1, RA2, , RA3, number of scholarships awarded, number of unique applicants awarded, the maximum and minimum total awards for a student. This is actually a separate stored procedure that is called, one for normalized and one for denormalized. I will use the phrase “Insert Into Analysis” within the seven algorithms descriptions to specific this section and will discuss this section after I dissect this part of the algorithm separately.

### Algorithm 1- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Declare @CurrentAmount

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Get Scholarship Applicants

Get Lowest Ranking for Current Scholarship Applicants

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @CurrentAmount

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 2- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Declare @CurrentAmount

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Get List of CurrentResults for Awarding Group

Get Scholarship Applicants Who Haven’t Exceeded Minimum Award

Get Lowest Ranking for Available Scholarship Applicants

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @CurrentAmount

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 3- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Declare @CurrentAmount

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Get List of CurrentResults for Awarding Group

Get Scholarship Applicants Whose Current awards + Scholarship Amount Haven’t Exceeded Minimum Award

Get Lowest Ranking for Available Scholarship Applicants

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @CurrentAmount

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 4- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Declare @CurrentAmount

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Get List of CurrentResults for Awarding Group

Get Scholarship Applicants Have not already been awarded

Get Lowest Ranking for Available Scholarship Applicants

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @CurrentAmount

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 5- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Declare @CurrentAmount

Declare @CurrentSplitAmount

Declare @CountOfApplicants

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Set @CountOfApplicants as Count of Applicants for @CurrentScholarship

Set @CurrentSplitAmount =@CurrentAmount/@CountOfApplicants

Get List of CurrentResults for Awarding Group

For each Current Applicant

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @ CurrentSplitAmount

Next Applicant

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 6- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Declare @CurrentAmount

Declare @CurrentSplitAmount

Declare @CountOfApplicants

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Set @CountOfApplicants as higher of the number of actual applicants or maximum applicants from the input parameters

Set @CurrentSplitAmount =@CurrentAmount/@CountOfApplicants

For each Current Applicant >= @CountOfApplicants (Ordered by Ranking Ascending)

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @ CurrentSplitAmount

Next Applicant

Next Scholarship in Looping Table

Insert into Analysis

### Algorithm 7- Basics

Get Input Parameters

Get Scholarship and place in a looping table

Set @CountOfScholarships to count of scholarships in looping table

Set @ScholarshipCounter to 1

Declare @ScholarshipWinner

Declare @CurrentScholarship

Remove previous results with these input parameters

Loop through each scholarship using @ScholarshipCounter

Declare @CurrentAmount

Declare @CurrentSplitAmount

Declare @CountOfApplicants

Declare @PeopleToDivideBy

Declare @ApplicantsWithMinimumAmounts

Set Scholarship as @CurrentScholarship with @ScholarshipCOunter pulling from looping table

Set Amount as @CurrentAmount with current scholarship amount

Set @ ApplicantsWithMinimumAmounts =@currentamount/@MinimumAward from input parameters

If @ ApplicantsWithMinimumAmounts < 1 then set @ ApplicantsWithMinimumAmounts=1

Set @CountOfApplicants as number of applicants for scholarships

Set @PeopleToDivideBy as lesser of @ ApplicantsWithMinimumAmounts or @CountOfApplicants

Set @CurrentSplitAmount =@CurrentAmount/@ PeopleToDivideBy

For each Current Applicant >= @ PeopleToDivideBy (Ordered by Ranking Ascending)

Set Applicant as @ScholarshipWinner

Set Result Into ResultTable with @ScholarshipWinner, @CurrentScholarship, and @ CurrentSplitAmount

Next Applicant

Next Scholarship in Looping Table

Insert into Analysis

### Insert Into Analysis

This algorithm poses a particular challenge to do in declarative fashion and uses aggregate grouping, Common Table Expressions (CTE), and window functions such as LEAD and LAG to determine the order of various elements. This could also be done in a looping fashion declaring all of the variables at the top of the loop and then calculating the values as they go through the loop. However, MS-SQL Server is optimized for CTE and windows functions so I choose to resolve this in a more declarative manner. However, explaining the algorithm needs to use higher level ideas such as grouping to understand how this algorithm works in my MS-SQL Version. The explanation also requires blocks that represent the CTE tables that serve as inputs to later CTE Tables. For this reason, this algorithm is shown in block form.

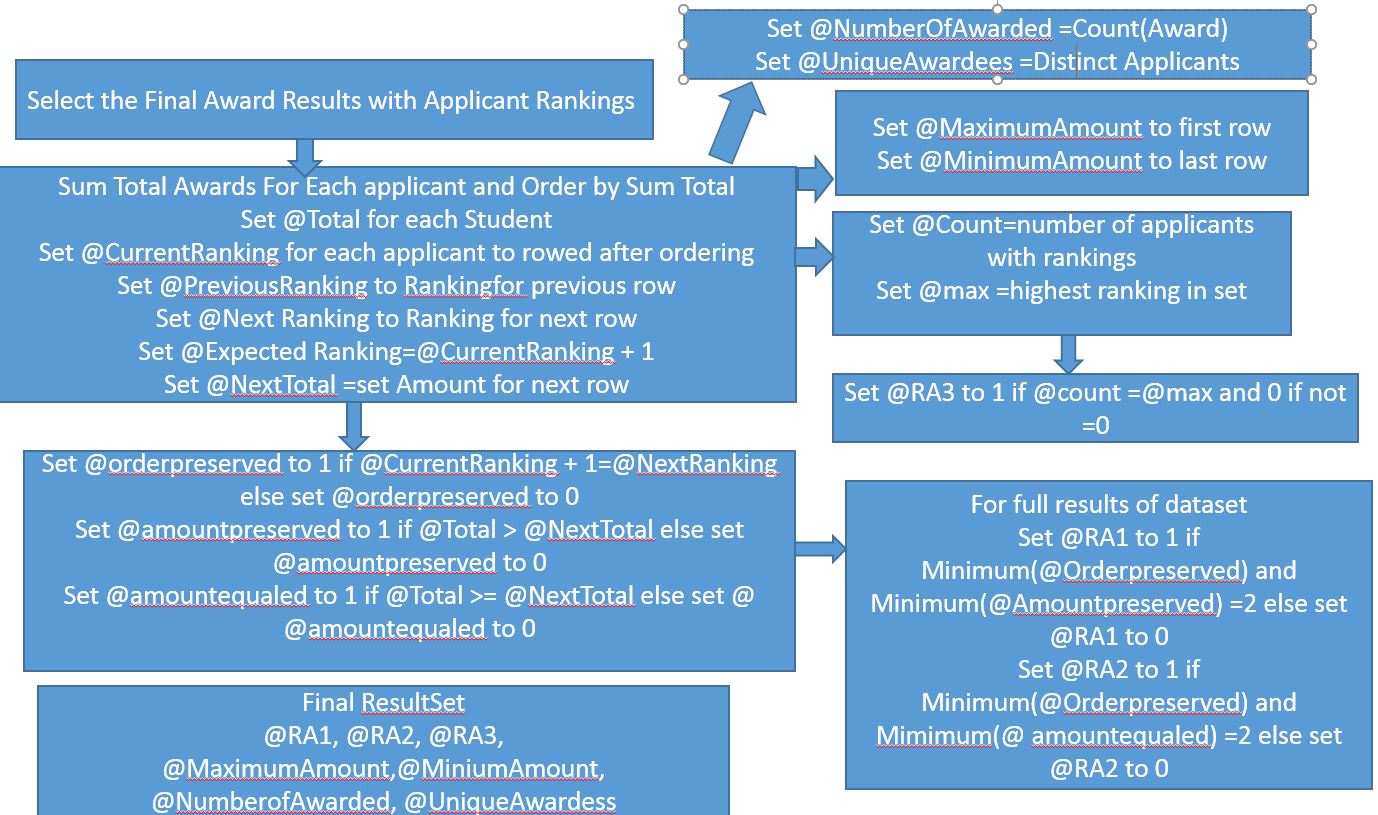


Figure 7 Insert Into Analysis Algorithm

# Appendix -Competencies

C1: Computational and analytic thinking and doing: Students will establish the ability to exercise the four key techniques of computational thinking: decomposition, pattern recognition, abstraction, and algorithms in solving information and data challenges, in addition to analytically.

C1.A: Decomposition: Students will be able to break down a complex problem or system into smaller, more solvable problems.

         The student broke down the 14 run algorithm stored procedures (7 normalized and 7 denormalized) into code for the algorithm and a separate analysis stored procedure which could be used in each of seven algorithms. The student also used a scheme to automatically determine whether the requested stored procedure should work on normalized or denormalized data and call the correct stored procedures.

 C1.B: Pattern recognition: Students will be trained to look for similarities among and within problems.

         The student was able to work with the similarities of code between the 7 algorithms to create one master procedure that runs all 7 at the same time.

C1.C: Abstraction: Students will gain the ability of recognizing and focusing on the essential components of a problem/issue while ignoring distracting peripheral factors in order to develop one solution that works for a class of problems.

         The student was able to recognize the important parts of the scholarship awarding process and abstract the relevant information for the meta-algorithm.

 C1.D: Algorithms: Students will be able to design and implement a step-by-step solution to a problem, including design and implement a computer algorithms using a computer language to solve a problem.

         The students use the MS-SQL language to solve the algorithm.

C1.E: Students will demonstrate fluency in at least one programming language.

* The student demonstrated fluency in the SQL programming language.

C2: Data manipulation, analysis, and interpretation: Students will obtain the skills of collecting, manipulating, and analyzing different types of data at different scales, and interpreting the results properly.

C2:A: Students will be able to identify specific types of data for different analytical methods

* The student created a normalized and denormalized data scheme that adhere to data standards. The student created an experiment to note the which different algorithms were considered more fair with different forms of data.

C2:B: Students will be able to use/develop efficient computational methods to clean, format, transfer, and store data.

* The student created a query to pull sample data from a live database.

C2:C: Students will be able to apply appropriate statistical, machine learning, visual analytics, and other techniques to identify patterns and make sound predictions with given data.

The student used graph representations to communicate the seven different algorithms. The student created graphs in R to describe the results of different results.

C2:D: Students will be able to develop methods to align and integrate data from multiple sources.

* Student created denormalized system to allow integration from diverse systems.

C2:E: Students will understand the ethical and legal requirements of data privacy and security.

* Student created queries from live database that keeps student’s private data anonymous.

C3: Communication and teamwork: Students will acquire skills to work with others within and across disciplines and be effective communicators.

* Student worked with others in Financial Aid to get ideas about different ways to award scholarships to large number of students.

C3.A: Students will acquire experience working in an interdisciplinary team, either as a productive team member or a team leader.  Students will become effective project managers.

* The student worked with Financial Aid professionals to devise the needs and create the proper algorithms.

C3:B: Students will be able to effectively articulate various evidence supporting a solution and to communicate the results of their work, using appropriate graphics, visualizations, multi-media vehicles, or artistic performance.

* The student used graphics and visualizations to communicate concepts for the topic in the paper.

C4: Creative contributions:  Through experiential learning, students will know how to conduct original and innovative work, involving computational thinking, data-intensive methodologies, and/or human-centered designs that will extend the body of knowledge in the field of Information.

* The student created an original work which used modular program to allow the process to be run from a single stored procedure.

C5: Ethics and Values: Students will demonstrate an understanding of information/data ethics, and the values of the information fields to serve diverse user groups.

* The student keep confidential student data from being unearthed and use private GIT Repository to keep data safe on previous work.

# Biblography

Belloni, Alexandre, Mitchell J Lovett, William Boulding, and Richard Staelin. 2018. “Optimal Admission and Scholarship Decisions: Choosing Customized Marketing Offers to Attract a Desirable Mix of Customers.” Accessed June 29. doi:10.1287/xxxx.0000.0000.

Biró, Péter. 2018. “Student Admissions in Hungary as Gale and Shapley Envisaged.” Accessed July 10. https://www.researchgate.net/profile/Peter\_Biro4/publication/237834167\_Student\_Admissions\_in\_Hungary\_as\_Gale\_and\_Shapley\_Envisaged/links/569ff89608ae21a5642727f0/Student-Admissions-in-Hungary-as-Gale-and-Shapley-Envisaged.pdf.

Codd, E F. 1990. *The Relational Model for Database Management : Version 2*. *Database*.

Gale, D and Shapley, L.S. 1960. “College Admissions And The Stability of Marriage.” http://www.dtic.mil/dtic/tr/fulltext/u2/251958.pdf.

Nobel Price Committee. 2017. “Stable Matching: Theory, Evidence and Practical Design.” Accessed August 28. http://www.nobelprize.org/nobel\_prizes/economic-sciences/laureates/2012/popular-economicsciences2012.pdf.

Rachmawati, Sekar Rizky. 2017. “UNIVERSITY OF OKLAHOMA GRADUATE COLLEGE SCHOLARSHIP-STUDENT MATCHING PROCESS OPTIMIZATION.” Oklahoma. https://shareok.org/bitstream/handle/11244/52949/2017\_Sekar\_Rizky\_Rachmawati\_Thesis.pdf?sequence=2.

Roth, and A E Sotomayor. 1989. “The College Admissions Problem Revisted.” *Econometrica* 57 (3). https://web.stanford.edu/~alroth/papers/1989\_E\_College\_Admissions.pdf.

Shapley, L, and A Roth. 2012. “Stable Matching: Theory, Evidence, and Practical Design.,” 5 pp.

Teo, Chung-Piaw, Jay Sethuraman, and Wee-Peng Tan. 2018. “Gale-Shapley Stable Marriage Problem Revisited: Strategic Issues and Applications.” Accessed July 10. https://pdfs.semanticscholar.org/6326/45fb5e7eb4e6c959fe42ed23d9a54cbb81d1.pdf.

The Royal Swedish Academy of Sciences. 2002. “The Prize in Economic Sciences 2002 - Press Release.” *Nobelprize.Org*. https://www.nobelprize.org/nobel\_prizes/economic-sciences/laureates/2012/press.html.