# 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 for many years. One of the most important matching problems that hasn’t been given much discussion is the scholarship awarding process. Matching students to scholarships is an important component of many schools plans to help their students and institutions succeed.

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. But 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. These include having higher ranked student receiving less than lower ranked students or no awards at all.

This paper will explain seven algorithms based on common scholarship committee’s practices that can be used in combination to try to minimize or remove these quirks to the scholarship system. These seven algorithms with three parameters can used to explore how to award scholarships in a fair way while still satisfying logical consistency in the process.

# Related Work

Matching and optimization algorithms have gotten lots of research.

# Problem to Be Solved

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. 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. These algorithms 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 would 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 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.

# Methods

## General Ideas of Seven 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 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 vale for the award 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

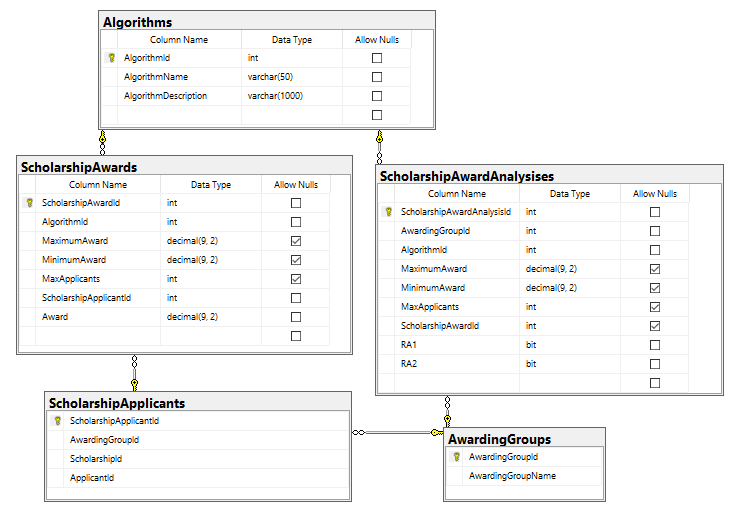
To implement these algorithms within the SQL language (MS-SQL Variant), it is required to create a data model. However, instead of only implementing the algorithm in a normalized model, I decided to create a denormalized model to allow easy import of data from a CSV import.

## 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.

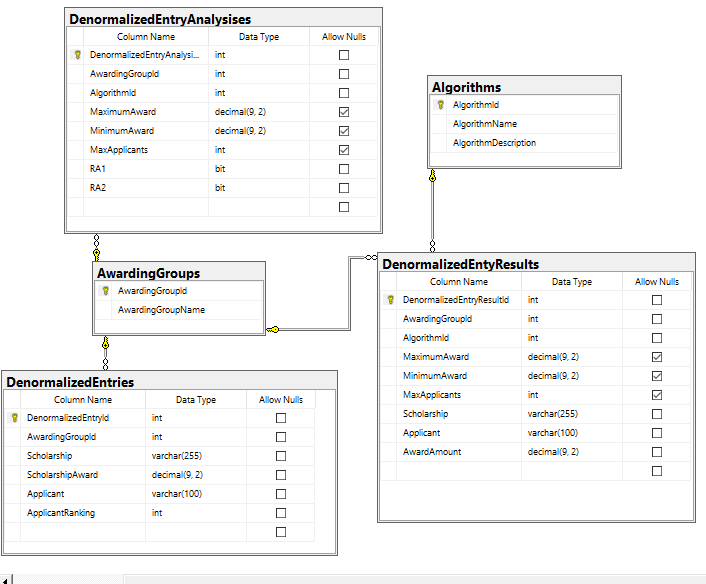


## Denormalized

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.



# Results

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>.

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.

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 query. For these reasons, I am including them as I will include them in my analysis of the algorithms. 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),

@MaxApplicants INT

Each algorithm ends with an input into an Analysis table which has calculated RA1, RA2, and RA3. The algorithm is the same for each algorithm. I will explore this algorithm separately later and will only include “Insert Into Analysis” within the algorithms descriptions for each.

So now I discuss how each algorithm works.

### 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 and window functions such as LEAD and LAG to determine the order of various elements.

Get Input Parameters

Delete From Current Analysis for Input Parameters

Set @grouptable =Group Join Scholarship Awards and Applicant Rankings with Input Parameters order by total award descending

For total for each applicant and total their award amount and give them a ranking and get the next ranking for the next row in the dataset, add 1 to get the expected ranking, and get the amount from the next row

For each row

set @orderperserved as 1 if next ranking and expected ranking are equal and zero if not

set @amountpreserved as 1 if total > next amount and 0 if not

set @amountequaled as 1 if total >= nextamount and 0 if not

if @orderpersrved and @amountperserved are both 1 then RA1 is true else false

if @orderperserved and @amountequaled are both 1 then RA 1 is true else false

Get maxranking from grouptable and get the count for grouptable where amount >0

If the maxranking and the count are equal then RA3 is true else false

# Discussion

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.

# Conclusion

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.

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 pushed through 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 orders 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 -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 a group of larger problems in the scholarship process into various modules that solve them.

 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.

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.

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.

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.