Fairness and Algorithmic Approaches to Project-Team Allocation

By Oliver Pinel

Abstract

This report delves into the fascinating world of algorithm design for team-to-project allocation in IFB398, with a focus on tackling the complex problem of optimization under multiple constraints. We'll explore the intricate puzzle of how to ensure every algorithm meets the project's unique requirements while remaining flexible and efficient. As we journey through this process, I'll share insights from my hands-on experience in devising and implementing algorithms, showcasing the trial and error that led to valuable breakthroughs. Along the way, we'll also highlight key research areas that have shaped my approach and will be vital in future developments, offering a glimpse into the dynamic relationship between theory and application.

The Allocation Problem

While the allocation process and overall goal of this project has already been discussed in great detail in previous reports, as the nature of this report will focus heavily on the mathematical side of using algorithms in relation to the allocation process defining a foundation for common understanding and to build off of will greatly simplify the research, concepts and exploration shared here. In addition, as our project has developed, our understanding of the allocation problem has changed, meaning that the original outline defined in earlier reports is outdated. To that end, I will be expanding here on the explanation of the problem outlined in our final team report, the Comprehensive Review of the Allocation Project, specifically in mathematical terms that will allow for the project to be explored in a new framework.

Definitions

Let us begin as we have before, by identifying the teams and projects. Given a selection of teams and projects, we can identify a given team as team i where i is an integer between 1 and m, and m is the number of teams. This should be noted as a different identifier than the given value TXXX assigned by the unit. Similarly, we can identify a project as project j where j is an integer between 1 and n, and n is the number of projects. Again, this is different to the value assigned by the unit as its identifier, PXXX.

The Problem

As we have already explored, the two semesters of IFB398 that ran in 2024 contained, in semester 1 just under 100 students and projects, and in semester 2 just under 50. When attempting to determine a stable matching between this, we must sort through potentially $100!$ different combinations. This is simply not possible to manage with raw data. To get a sense of the scale of this, I've generated two figures below.

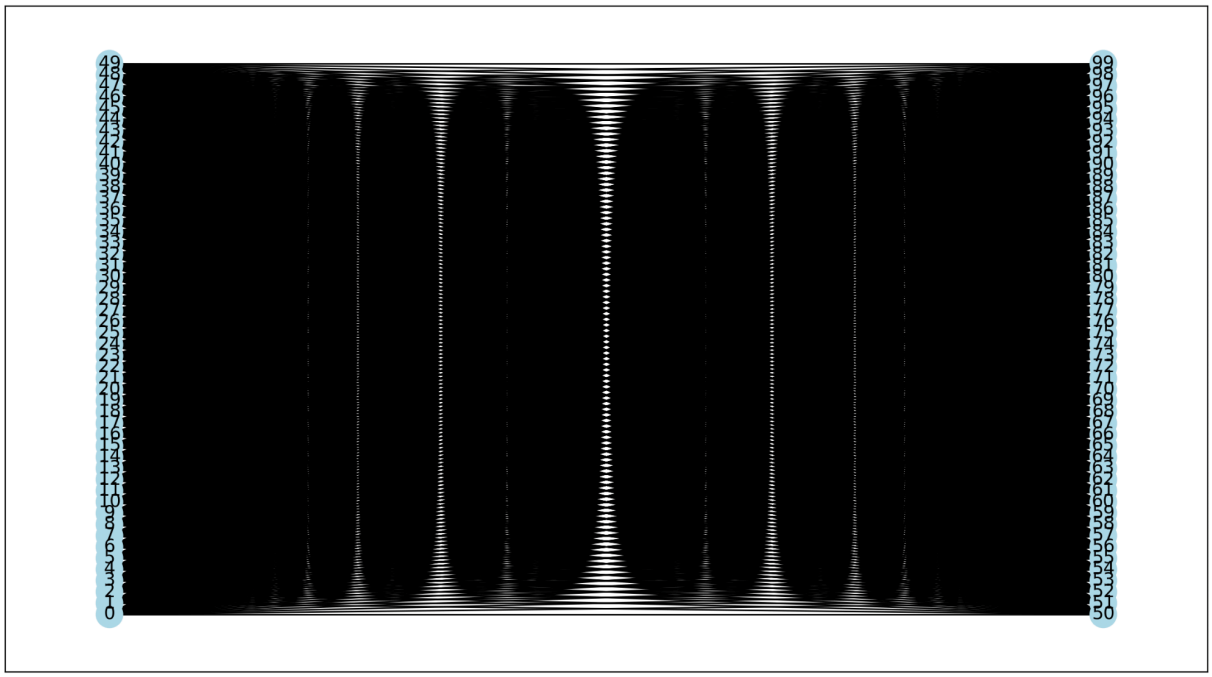


Figure 1: A Bipartite Matching of 50 Teams to 50 Projects

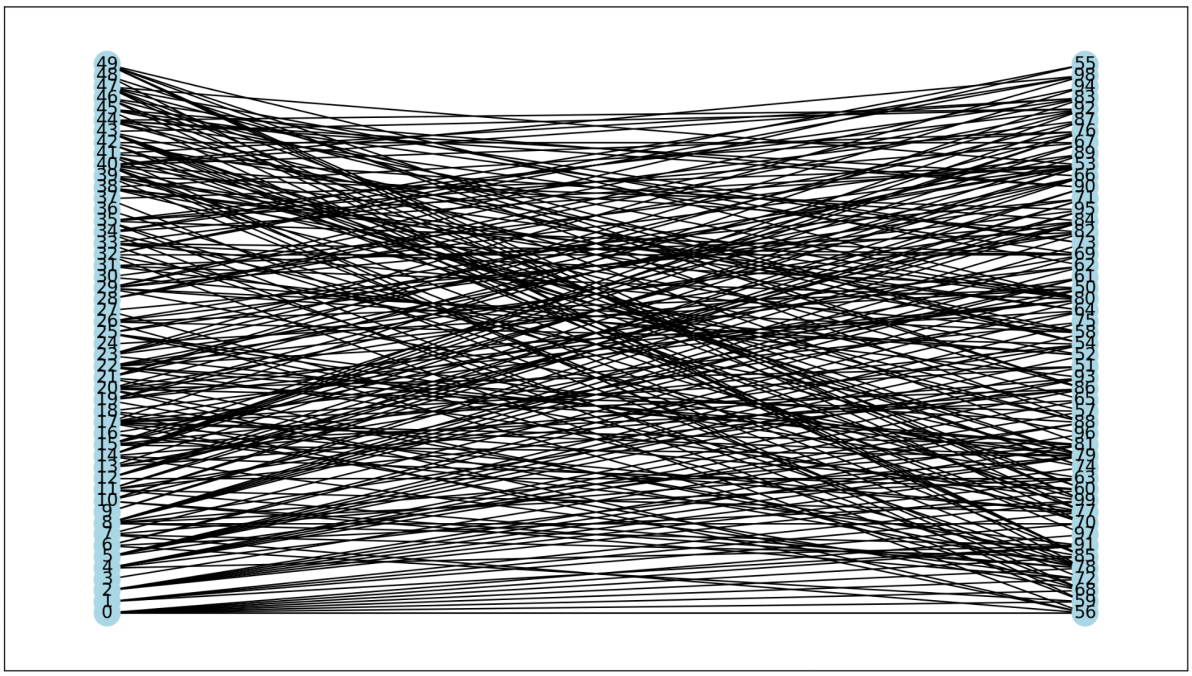


Figure 2: A Bipartite Matching of 50 Teams to 50 Projects with a 10% fit rate

As can clearly be seen, this visualisation is unreadable, and provides no benefit in actually determining which project should be allocated to which team. Even when incorporating the

newly applied impact scalar that can divide the problem into three distinct groups of teams and projects, it remains a large set of comparisons that, while not impossible to do by hand, would take far longer than is reasonably available.

But the research into displaying, representing and sorting data is not a main focus of this specific report, being covered by my team members. While their work will be important and certainly mentioned here, our focus will be on algorithmic means of reducing the sample size, optimising the choice range. This must be done carefully such that the fairness and integrity is not violated by our algorithm. There are a number of considerations that have been identified already, and when implementing an algorithm, it becomes a balance between them.

Thus it follows that the most common use case of an algorithm will be some form of ranking, of comparing, and determining a best fit. It will involve comparing two or more possible pairings, and identifying which one is ‘better’ to some criteria. Perhaps an algorithm can present the best choices, or sort a visualization based on the impact an allocation would have on the remaining pool of pairings. The goal of any algorithm should be to reduce both the complexity of the problem, as well as the time taken to do so. This in and of itself is simple. The challenge lies in achieving this without violating any of the other requirements, a much more involved goal.

This report will explore the different considerations, how they impact the allocation process and where an algorithm can be used in conjunction without violating the requirements. We will attempt to implement some concepts discovered through research, and explore the process as it is done. Throughout this, the report will develop a framework that can be used to determine where algorithms can be used to improve the allocation process, and how they can be properly justified.

What is ‘Fairness’ and how does it matter?

Something that has been extensively remarked upon in previous reports is the importance of ‘fairness’ when allocating teams to projects. It is particularly important that any decisions can be justified, and fair, to the students and industry partners who will have to work with the changes. But in this situation, what would be ‘fair’?

There are several metrics we can consider. Perhaps every team should get the best allocation so the overall matching is optimal in terms of the b value. This was our initial thought, and on the surface would certainly seem to be fair, giving equally harsh consideration on every team. By considering the common good and making the optimal choice based purely on numerical calculations, the sacrifice of some teams not getting their most optimal project meaning there is no pairing that is poorly allocated.

This purely mathematical algorithmic approach is what our team initially believed in, the ruthless but fair algorithm with no bias. And in our attempts to explore and research through the first semester, we implemented several different versions of these algorithms and found that they were quite successful. In a vacuum, where the project was narrowed down to purely numerical

values, this approach was certainly equally unfair to each team and produced mathematically optimal allocation sets to what was actually implemented for the subjects. Despite this, the method is almost worthless in practice for two reasons, one of which will be explained here.

The b values, along with the associated values used to calculate them turned out to be less accurate measures than needed to properly compare teams. While when only considering one team and several projects they were quite effective in comparing the differences, measures between teams proved more complex owing to the differences in available data. As such, any algorithm based purely on the b values could not be entirely relied upon. Equivalently, there was no other data to work with, and so any algorithm would have to be designed knowing of this floor.

Perhaps a different definition of fairness could be useful. Maybe the students who formed a team themselves could be allocated first, and the teams formed by the teaching staff could be done second? This could arguably provide better results, as self formed teams would likely have a better idea of what they wanted to work on and possibly have a similar overlap in specialties. It would also incentivise and reward proactive students and teams. Of course, while I personally know a student who was allocated a team and project because they couldn't be bothered to make those connections themselves, and likely cared less than most students about what they worked on, I also know two students who couldn't form a group due to circumstances beyond their control. It certainly wouldn't be fair to penalize them for that.

Mayhaps extra weighting could be applied to teams that put in preferences, or even increase the default preference scaling from 0.1 to some higher value. Implementing functionality to allow for that was one of our better successes in the experimental system. Maybe a deterministic algorithm could be used to create the order in which teams are allocated, instead of doing it one by one randomly, although that topic will certainly cross over with minimising greedy allocation, as will all fairness considerations.

Algorithms could be used to present the best choices for each team, or identify pairings that are good whilst also not impacting the pool for others. They could filter out the bottom half of options that are the worst fit. They could separate the potential pairings into the three impact ranges and do them one by one. Allocations could start from the highest average b value to lowest, or the ones with least impact on other teams. All of these could be argued to be fair in one way, and yet be unfair in others, even if it's not clearly obvious.

In a presentation in 2018, Arvind Narayanan, a professor of computer science at Princeton University, argued that there are 21 different definitions of fairness. And that while individually all of them are fair they also contradict themselves. It would certainly be fair to be equally fair to each group in giving them their best pairing possible, just as it would be equally fair to be uncaring to each group and produce the best overall pairings, even if that means some teams get worse options.

In short, the fact is that fairness is arbitrary. With many valid yet conflicting definitions it is impossible to meet all criteria simultaneously. The unit coordinator, and the team that is developing the system must ultimately choose one of the many subjective definitions of fairness that can be best defended in retrospect to the people impacted by the decisions.

Fairness within our constraints

As we've explored many times before, there are a number of requirements that must be adhered to when doing the allocations. It is essential that there is a human factor, a human in the loop when assigning the pairings. This is needed so not only can the entire allocation system be defended as fair, but so that each individual allocation is also something that can be justified. It is more important that every team gets both a project they can complete and that the allocation can be defended as fair, than each team getting the best project for the overall greater good.

To that end, we must consider how algorithms can be used within this requirement. The most important understanding I have come across was a phrase from a report called "Fairness in Resource Allocation". It states that algorithms balance an exchange between fairness and efficiency. In the case of our project the "efficiency" of an algorithm has been greatly explored, defining what a good result is. And as such, when implementing an algorithm for this project, there must be a choice between how much it can improve the efficiency of the system, be that time taken or overall average b value of the allocations, and whether the algorithm being used will make the allocation process less 'fair', less defendable to the students as equal opportunity and consideration for everyone.

How can algorithms be used?

Now that I have clearly outlined the pitfalls in implementing a 'best' algorithm to fix an obviously unfair situation, let us explore in more depth what specific ways algorithms can improve the allocation process.

Our team originally found the most obvious situation for an algorithm to be in finding a best fit matching, as it admittedly took quite a while for us to fully understand the level of complexity in balancing efficiency versus fairness. We focused entirely on that and ignored the necessity in other places.

Truthly though, the matching algorithm is possibly the least necessary implementation of any kind of algorithmic assistance that is useful in the allocation process. There are a variety of other areas that could be more useful to have some algorithmic assistance. The goal of an algorithm or computational assistance in sorting through data and making recommendations is to reduce the time required for a human, as well as identifying the best choices. This can work very well in conjunction with the research Raphael did into representing the datasets, and comparing data in readable forms, unlike what is seen in Figures 1 and 2. By combining an

algorithm to help filter the data in a specific way, new insight into the remaining teams can be found.

Additionally, algorithms can be particularly used for very specific purposes. Some that I have been considering after the results of the experiment are different ways to identify and present data. Particularly, identifying overlap between teams to separate into further subgroups the teams that have a small overlap between potential projects, an example that will be explored further on.

Finally, and possibly most importantly, there are ways of measuring fairness mathematically with algorithms. An implementation of a MAD or SAD score, as outlined in the Fairness report referenced earlier, could measure whether any team was given an unfair outcome overall and would potentially need revisions.

The Algorithm Contradiction

The algorithm contradiction - the most essential thing to understand about any form of algorithmic or computational implementation.

This took me quite a while to properly comprehend, and the idea that my perfect algorithm which provided the best matching allocations possible wasn't desirable was a something I wasn't ready to accept until the experiment. The contradiction comes from two fundamental facts about the allocation process that need to be recognised and accepted.

1. There are too many potential pairings and matchings to sort through without an algorithm
2. Any and every decision must be made by a human with no influence from an algorithm

These two requirements are fundamentally in conflict with each other. We need some kind of algorithm to sort, filter and contrast the data, and the algorithm cannot be good enough at it's job that the human loses any agency. Every decision, whether it be how the data is filtered and sorted, or whether a specific team gets allocated a project over another, must be done without any computer suggesting one is better, at least not obviously.

We need an algorithm, and it can't be perfectly efficiency.

Why is a Perfect Algorithm a bad idea?

Even as I outlined in the previous section, it can still be difficult to understand why a good algorithm is bad. That is because this needs to be held in conjunction with the understanding of the balance of an algorithm, the exchange between efficiency and fairness. It is from these two understandings, the algorithm contradiction and the exchange of efficiency for fairness, that we can devise a framework for identifying where an algorithm can be implemented.

Even so, there should be no problem with a perfect algorithm, one might think. I certainly did, even understanding this. As long as it doesn't make any decisions, and simply presents options in data, it can't reduce the fairness just by existing as an option.

And yet, the experiment proved the exact opposite. Just by existing, the algorithm presented an option. Simply by being seen as a presentation of data, the user was influenced into following its choices, both by the perceived accuracy along with being simple to access and read. In fact most allocations were chosen simply because they were presented by the algorithm, and when looking into them specifically, they looked good enough.

And perhaps they were the best choices, perhaps overall they would have produced the most benefit, the most efficiency of the process into the system, but because there was such a heavy influence from an algorithm, the results were suspect, and could not be properly defended.

The algorithm had too much influence over the system, something very visible from watching the recording of the experiment, which I highly encourage. Any algorithm would have to be much more subtle, or at least not present its options so selectively, without exposing a range, giving more weight to the user.

Identifying where an algorithm can be useful

- Balance of algorithm, explore more
- Have notes elsewhere can't remember where its from

So, we've developed and understanding of the use cases of algorithms, as well as an understanding of what must be considered when choosing where and how to implement one. Now, we can begin to develop an argument to consider a situation and work out whether an algorithm is suitable.

The situations we have explored so far in this report are all consistent in one factor, in that they consider replacing some human action with an algorithm. The goal is to optimise some action, replacing it with a programmatic process, and so the best way to begin is to consider every single aspect of the allocation process, and determine by some metric whether it can be replaced.

Despite the seemingly polarised balance between efficiency and fairness, it is not a zero-sum game. Improving efficiency does not have to reduce fairness, a key insight that will allow for two rules to be devised.

We can begin by considering the things an algorithm can do versus things that they cannot, at least not without violating our contradiction.

An algorithm can choose a deterministic formula, identify groups, identify overlap, produce a list of options, or even decide to show a grouping of projects for a specific team, given that while doing this it is not selectively producing a bias. An algorithm cannot selectively show data, or

abuse visualisations to influence a choice, nor can it produce a ‘correct’ option, ignoring other choices.

In short, an algorithm can do anything to improve efficiency as long as it does not reduce fairness. Again, it’s not a zero-sum game, and as such, algorithms can be implemented without reducing fairness, or indeed increasing it.

If a situation or aspect of the process is already unfair, or if introducing an algorithm will not reduce the fairness of the overall allocations, then it is acceptable. If the situation is already considered fair by whatever metric is chosen, then an algorithm cannot be introduced to improve the efficiency unless it can be proven to have no impact on fairness.

For the consequences of this project, it is much more important that fairness is retained than efficiency is improved, and thus we can identify a priority and balance to consider.

With this situational guide, along with the two main components of an algorithm in this system it is very possible to properly research some concepts that can be used to implement an algorithm, as well as identifying when it's appropriate.

Exploration of Research

One important thing to keep in mind when implementing any kind of algorithm to show data in a certain way, is that simply by presenting data a narrative is produced that will influence the user. Humans are more likely to read information in a certain order, and whether that is intentional or not, it will influence the user. An algorithm must consider the order in which it shows data, and whether that would be fair. The same data, simply shown in different orders, is likely to result in different conclusions being drawn. Selectively grouping data to present an option, for instance showing the top three choices for one algorithm? This is likely to convince the user that even if the fourth option is close to the others, its exclusion from the presented choices will automatically make it less valuable than the first three. This issue was explored in a study by Fox, Ratner, and Lieb, which examined how subjective grouping influences choice and allocation (Fox et al.).

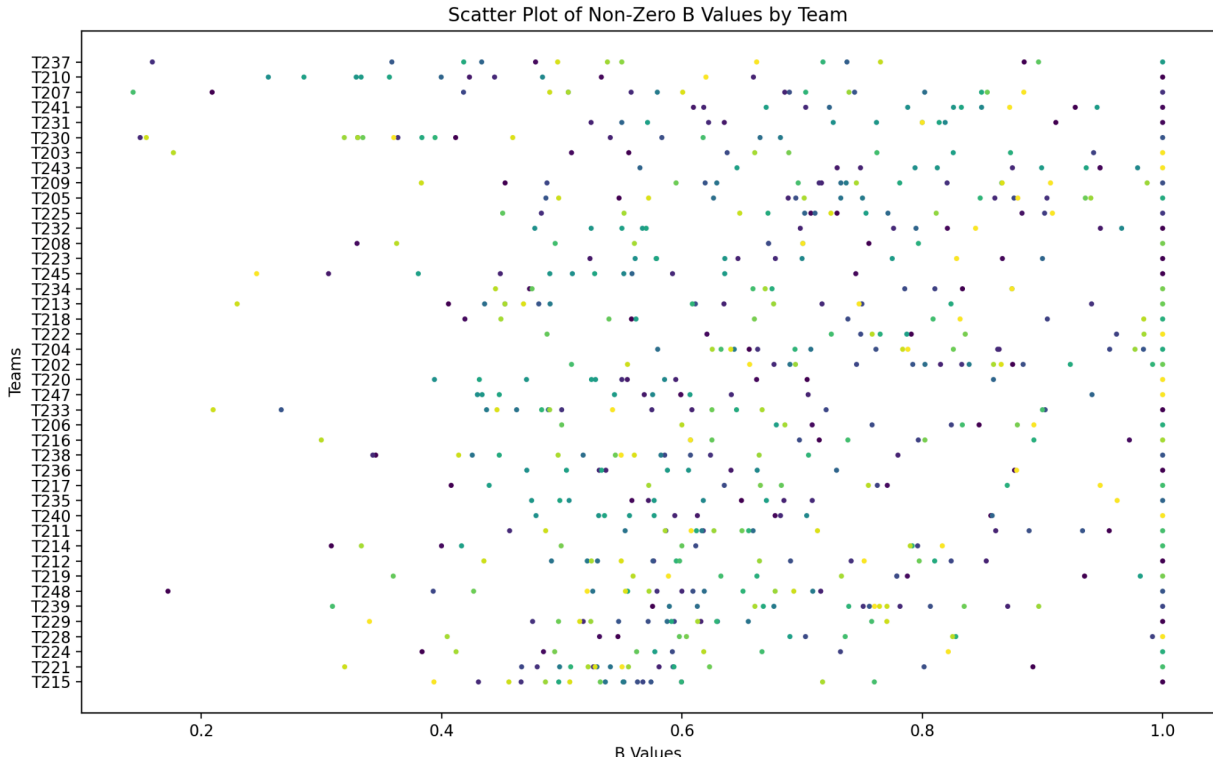
Mentioned earlier was a MAD and SAD score. These are explored in the Fairness in Resource Allocation report, but further detail is built on them with the introduction of the Gini index, which provides a measure for scoring inequality. It could be modified and utilised as a metric throughout the allocation process, as it records a score comparing each element, in this case each team, against each other as to how well they are balanced fairly. It could provide an indication as to whether bias is occurring during the process of the allocation, as a higher score means that one team is being favoured over the others. Similarly, the report also introduces Jain’s index, which measures the inverse covariance between the teams. This could be produced alongside the Gini index, as it is a measure of how the overall satisfaction between each team's allocations varies.

A report written by Christian Hass on Two-Sided Matching with Indifferences, presents two equations for calculating the fairness of a matching. Firstly, he argues that a measure of satisfaction for all participants can be measured as the averaged matched rank for all participants. In our case, this would involve determining a score for the average b value of each allocation for every team at the conclusion. In contrast to this, Hass also presents an equation for fairness, wherein he considers the difference in average between every allocation. This can be used to identify any significant outliers, as opposed to getting a general sense of the inequality. By combining the two indexes, it could be possible to quickly improve the allocation set by changing only a couple of allocations around.

Of course, in order to compare teams, we need some method of making them unitless, or at least to the same scale. One method of achieving this would be to normalise each team. There are several normalisations possible, but in order to make each team comparable, they should be normalised to their maximum value. To prove this, let us consider the graph of the teams vs b value for each project that they have an impact value of 1 for.

Figure 3: Scatter plot of Non-Zero B Values sorted by maximum value

As can be clearly seen from this, there is no way to compare one team to another, as they operate on different scales. Any form of comparison, be it highest value, or biggest range, will in some way be less fair to other teams. As such, we can begin by normalising each team to itself. As explored in the report by Chris Tofallis, “Add or Multiply? A Tutorial on Ranking and Choosing with Multiple Criteria”, we can use this to create a common scaling between teams, so that they can be appropriately compared to each other. This form of restructuring of the data is essential, as it, according to our previous conditions, reduces unfairness by measuring each team only against themselves, and thus when they are compared the highest possible pairing for a low scoring team will not seem subpart o a team with many high scoring pairings. The result of that normalisation can be found below.



From this, we can clearly see that every team has their own fit of good options, now moved onto a similar scale. From this, we can begin to compare their distribution or other datasets, as explored in the article How to compare two distributions in practice, by Alex Kim. An example of attempting to apply this can be found in the following section.

Determining the order of teams to be allocated

This section will cover one section of research and an attempt to implement an algorithm, according to the requirements we have devised above.

When considering the allocation process, disregarding producing an algorithmically optimal allocation set, one clear improvement could come from creating a system for determining the order in which each team is allocated a project. It should be noted that this specific instance has quite a heavy overlap with the research done into greedy allocation, and in particular was

chosen because of this. Hopefully, this should demonstrate an overlap with that topic, as well as an application of the ideas found here, and an inclusion of some of the research found in my research notes.

We must begin, as outlined in the devised framework by determining whether it is appropriate to replace the current working with an algorithm. As was seen in the semester 2 allocation recording, there is currently no pattern in the way each team is chosen, rather whatever feels right at the moment. Whilst it could be argued that this is in and of itself a fair method by not directly favouring a specific team, the greedy nature will mean that the later teams are restricted in their pool of options. Thus, we can definitively say that the current method is not fair, and can begin looking for a way to improve efficiency, and potentially increase fairness.

One method of doing this is with a deterministic comparison between each team, in order to identify a priority ranking. From the previous section, we can see in figure 4 that our data can be approximated into equally normalised data. From this, made the assumption that the data could be treated as an approximate normal distribution. This would allow for measures of mean and standard deviation to be calculated for each team's potential options, giving a way to compare them. Particular care had to be taken when comparing, as outlined in the Add or Multiply reports linked in the research, as determining which makes sense in the context of the mathematics is essential.

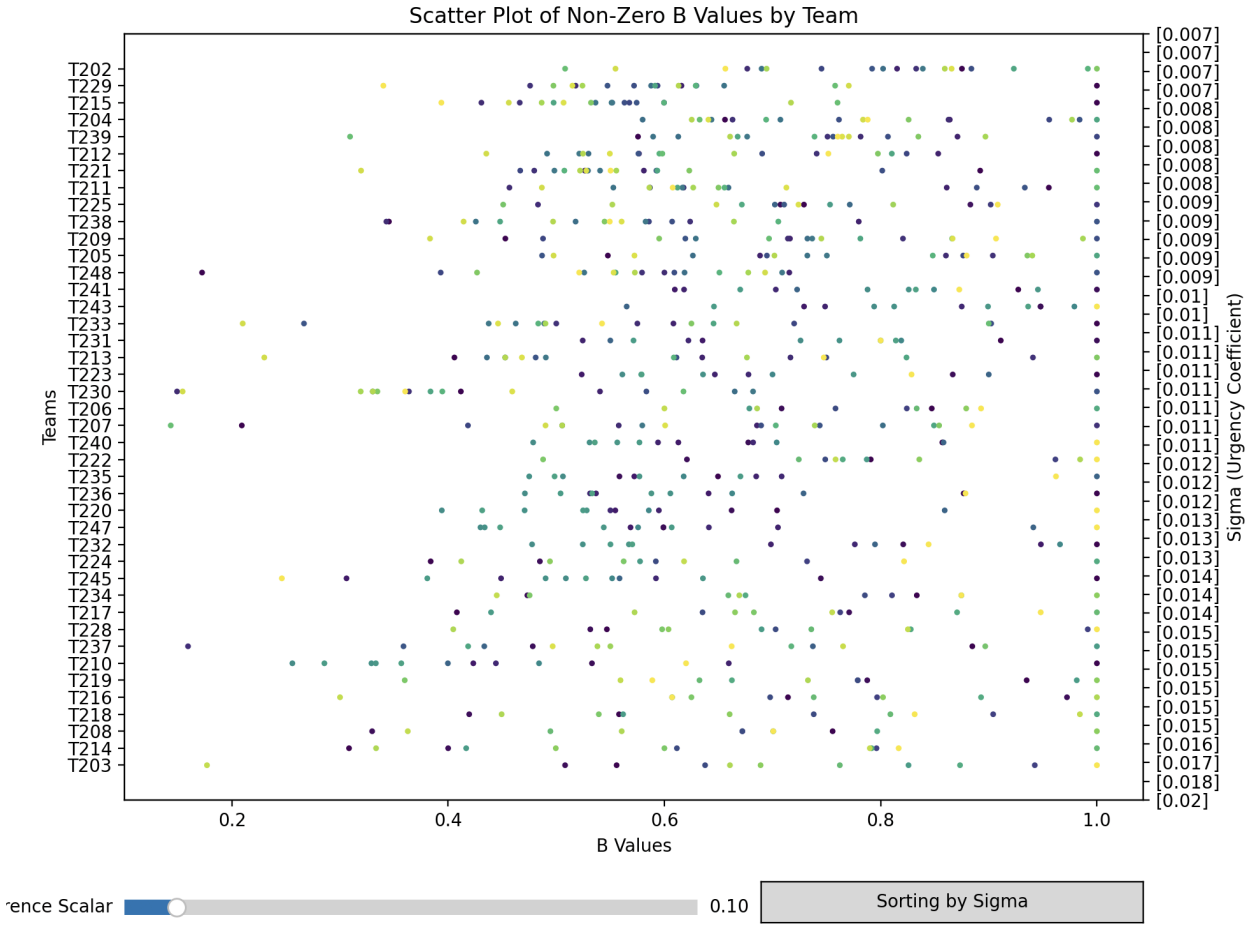
So, the goal of this was to determine a deterministic formula that would produce a characteristic measure of each team based on the available projects that would allow them to be ranked. To accomplish this, I began by considering what factors would impact the order in which the team had to be prioritised. Perhaps whether they put in a preference at all would mean they should be given a preference themselves. How many projects did they have that they could be allocated to? Mayhaps the higher mean for the normalised values meant that they had a better fit with their projects. But did that mean that they should be done first or last? Does having a wide spread in the normalised range mean it was more important or less to do them first?

While considering this, it was instrumental to develop some code to implement a shifting display. This can be found in the python script main.py, and gives demonstration of how the spread can change as the preference was most prioritised. From this, it became clear that there was no physical meaning to the mean of each distribution, at least in the sense that it could be allowed to prioritise one or the other. In fact, the spread was a much better measure. Based on this, I devised the following hypothesis as a characteristic to measure each team.

$$\sigma_i = \frac{standard\ deviation}{number\ of\ projects}$$

Where each team can be given a score sigma based on how spread out each team is. This score is scaled by how many options they have. As such, given a large spread and a small number of projects, sigma would be higher and would possibly need to be priorities to make sure that their small range of good options is done sooner. Similarly, a team with a small spread and a large number of projects is likely not as urgent to be done.

Is this a valid measurement? It was derived using research but the actual formula came only from my studying the distribution of data. While that means it is not backed by other people's research, it doesn't necessarily mean it is incorrect. This is one of the cases where it could be an arbitrary acceptance of a method that is fair, and justifiable, simply because of how the data interacts. And studying a scatter plot sorted by sigma descending, we can see the trend makes sense, where the bottom teams should likely be prioritised due to their lacking options.



And so the derived coefficient could certainly be a fair argument for deciding which teams should be allocated projects first.

From there I began to consider a way to divide projects by how much overlap they would have with other teams. After some consideration of whether to add or multiply (a very important factor in making sure any coefficients are valid in context!), I devised another formula. For one team:

Let team 1 have the set of projects $A = \{p_1, p_2, p_4, \ldots, p_m\} \subset \{p_1, p_2, p_3, p_4, \ldots, p_n\} = P$

Let team 2 have the set of projects $B = \{p_1, p_2, p_3, p_5, \ldots, p_n\} \subset P$

Let P_i be the set of projects for T_i. $N_j(p)$ is the number of teams excluding T_i that rank project $p \in P_i$ in their top half. Thus,

$$\lambda = \frac{\sum_{p \in P_i} N_j(p)}{|P_i|}$$

In theory, this lambda value should convey a sense of overlap between teams, as a measure of how little impact one team being allocated a project would have on the remaining pool. One topic we have continually discussed over the semester is the idea of determining which teams impact the best fit of the allocation set the least. In the sense that if we were to change the projects allocated to that team out of their best options, would it have the least impact on the overall score. This lambda coefficient was a rough attempt to derive that.

Of course to determine whether it actually represented any value, it was essential to tie this meaningless unitless value to some visualisation. One topic discussed between us in our research has been the idea of network diagrams, showing the overlap between teams. The python script graph_theory2.py, displays a network graph showing the connections between each team given that the top half of each team's projects have an 80% or more overlap. It can be seen in the figure below

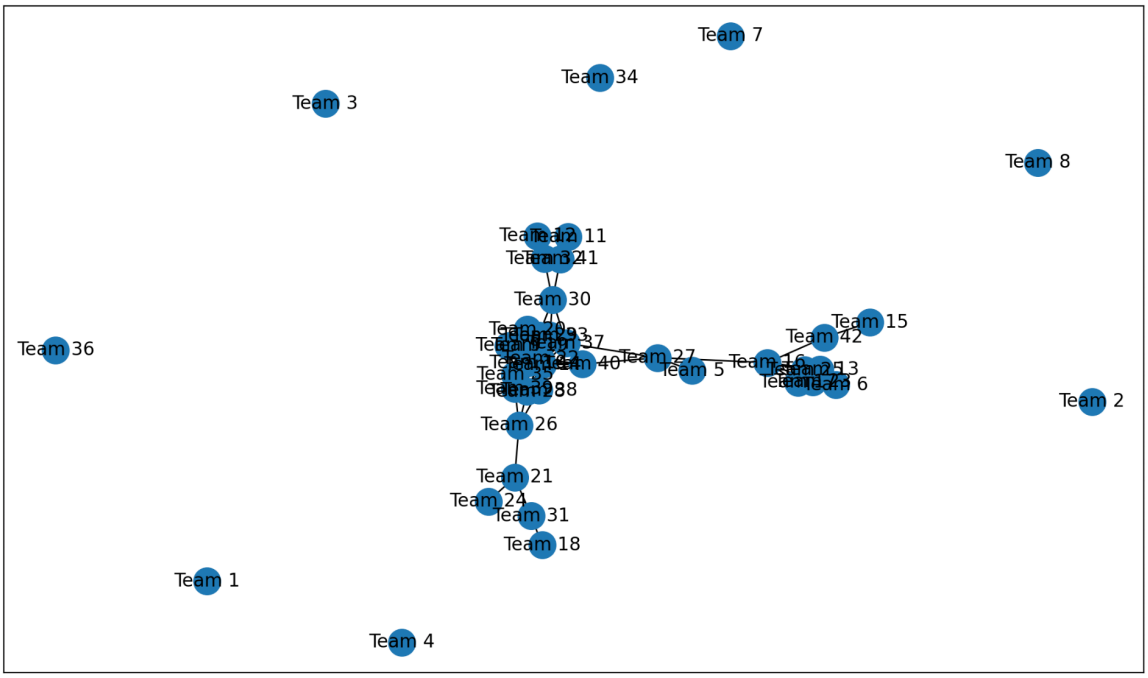


Figure 6: Network graph of 80% overlap between top half of best projects

Just from this visualisation of the data, we can clearly see some teams which have little impact on each other, as well as some small groups. But, as interesting as this diagram is, it has less relevance on its own to algorithms. In order to test the derived constant, I created this figure, including the lambda value of each node.

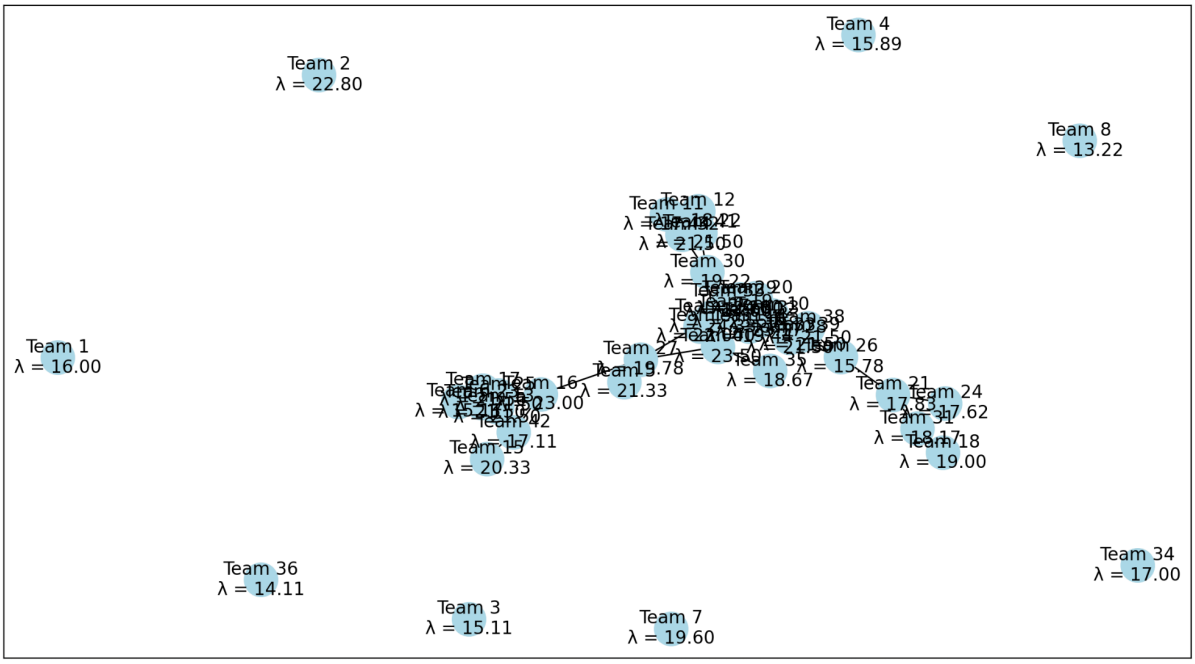


Figure 7: Network graph with 80% threshold and lambda values

It's immediately obvious that there is no correlation between the lambda values and number of connections. Perhaps there is potential in this lambda score, but currently it represents nothing of significance. As such, I had to discard the lambda coefficient as I did not have enough time to keep exploring this.

Recommendations and things to keep in mind

You've read this report. You've read our other reports, on research, experiments and the development process. Now comes the difficulty in determining how best to solve the allocation problem.

Not all of my research was summarised here. Many of the ideas I explored have been left as notes in order to keep this report to a somewhat reasonable length. There were several topics unmentioned, or only briefly explored that have potential, and my thought process can be found in the accompanying pdf Algorithms in Allocation. This follows both my research directions, a collection of sources that after carefully pruning were the ones I read through that had value. Personally, I would have liked to give more time to explore reframing the entire allocation process as a multi criteria decision making algorithm, as I believe that is the greatest potential in

working with allocating teams. Even now though, I struggle with the idea that algorithms on their own are not a viable solution to this problem.

To make a good system, algorithms will be needed in some form, even if it's to simply reformat the data for different visualisations. The potential combinations that can be examined are absurd and beyond human comprehension. But algorithms alone aren't enough. When developing a system, try to avoid the mistake our team made of dividing the project in two, a backend and frontend. Algorithms work best when in conjunction with different visualisations, not purely as a means to an end themselves.

Attached in this folder are several script files, used to generate the figures seen in this, as well as the development of the attempted ordering of teams. I've tried to create multiple files that build off of each other, instead of presenting a 'finished' solution, in the hopes that it can help show the work flow of trying ideas.

This project has great potential to involve algorithms to truly improve the allocation process, but it must be done delicately and with care, as to not overwhelm the human running the system. The most important aspect of the allocation process is the fairness, and any algorithmic implementation is better off being discarded than violating that, as such an aspect of a system could not be used.

There are numerous ways algorithms can be implemented into an allocation system, and most of them will rely on the way the system user interface is designed, or how data is intended to be shown to the user. Keep this framework in mind, and use it to enhance the user experience without making it any less 'fair', whatever measure of fairness is decided upon.

I personally found this aspect of the allocation fascinating, and really enjoyed the way it challenged my preconceived notions of how useful algorithms can be. Perhaps it was my mathematics background, but I really believe that every aspect of this process can be improved with an algorithm somehow. While an improved UI would certainly meet some of the other requirements, an algorithm is the most essential aspect of this, and balancing that against the need for fairness was an intriguing challenge.

Begin by determining an arbitrary limitation or definition of fairness and then work from there. Decide how the system will be fair, according to what definition, and it should be simple to work to make algorithms that benefit that specifically, to improve the efficiency whilst preserving fairness.