Two approaches

1. Research algorithms and find out how they could be applied
2. Identify facets of the allocation process that can be improved by algorithms

I have a system I'm designing that involves allocating projects to teams, based on how well they fit. Currently, we have done a test that involves providing an algorithm that finds the current best match out of the options and suggests it to the user, while they use other UI elements to sort, filter, compare and make allocations. There is one main requirement of the system that must be adhered to:

Allocations must be done by a human, and as such any decisions can be justified by the human doing it.

In interest of fairness, each allocation has to be made by the human user. The problem was the algorithm was so accurate and helpful, during the test it was solely used and its suggestions were the overall outcome. As such, I need to design algorithms that work more subtly, identifying key pairings, or providing small suggestions. I've decided to research this from two directions:

Research algorithms and find out how they could be applied
Identify facets of the allocation process that can be improved by algorithms

2. Identify Facets of Allocation That Can Be Improved by Algorithms:

1. "Factors influencing optimal project-to-team allocations"
2. "Challenges in fair project allocation decision-making"
3. "Weighting criteria in human-in-the-loop allocation algorithms"
4. "Avoiding biases in project-team allocations"
5. "Identifying hard-to-allocate teams in project management"
6. "Optimizing suggestions for reducing conflicts in team allocations"
7. "Human-centered constraints in project assignment algorithms"
8. "Designing adaptive algorithms for fair team-to-project allocations"
9. "Real-time feedback on team-project fit during allocation"
10. "Minimizing impact of team-project allocation on future decisions"

1. **Least Impact Algorithm**: Identify pairings that least reduce the quality of remaining options for other teams, based on similarity scores between teams and projects.
2. **Top-K Suggestions Algorithm**: Provide the human user with the top 3-5 best team-project pairings based on multiple metrics, letting them choose from these limited but optimized suggestions.

3. ~~**Feature Importance Algorithm**: Highlight which features (skills, location, etc.) of teams and projects are most critical in making a decision, allowing the user to prioritize those factors.~~
4. ~~**Multi-Objective Weighting**: Use an algorithm that dynamically adjusts weights for factors like cost, team skill, and project difficulty based on the user's manual decisions so far.~~
5. ~~**Cluster-Based Suggestions**: Suggest teams that fit within specific clusters of skills or project requirements, keeping the pool broader and less specific than a single best suggestion.~~
6. **Diversity Maximization**: Suggest allocations that maximize diversity in project-team pairings, ensuring a variety of team skills across projects, while letting users adjust the final selection.
7. ~~**Threshold-Based Filtering**: Automatically filter out options below a certain threshold of suitability, leaving only closely matched projects to choose from.~~
8. ~~**Historical Allocation Learning**: Suggest team-project pairings based on historical data from previous allocations, learning from user decisions in similar scenarios.~~
9. **Conflict Resolution Algorithm**: Highlight and suggest resolutions for conflicts between teams' preferences or project needs, without directly allocating the best option.
10. **Fairness-Based Balancer**: Provide subtle guidance on balancing team satisfaction across multiple projects, ensuring no single team ends up with significantly worse allocations than others.

<u>Algorithm should be used in determining the order of teams allocated</u>
- Each team is still chosen a project by person
- Makes the process less biased to whatever order is chosen
- Rigorous justification as to the order
- Has to be some order - may as well make it deterministic and 'fair' according to some criteria

Normalise b values
<u>Dont know what the best way to rank the data is so provide the options to the user</u>
Do it like a drought? I.e. rank the teams by how bad their available options are? Also normalised?
Allocate by how bad their available range is, like patient in hospital prioritisation
Algorithm used to create the order in which teams are allocated
- Ranking of priorities
    - Preferences input?
    - Available 'good' options?
    - Team details

Algorithm that identifies how much an allocation will impact the remaining pool

Minimising algorithm function thing - find pairings that have lowest overall impact on allocation score.

Normalise pairings for each team, then calculate distribution range on a normal distribution?
I.e. we have a set of projects P, a set of teams T and a set of allocations A = T x P. For each T_i, i \E 1,...m (m teams) we can normalise the values, and then assume normal distribution T_i ~ N(mu_i, sigma_i). Compare them somehow, not just expected but rank every T?
I.e. big mean small sd is low risk, small mean small sd is big risk, small mean big sd is big? Risk, big mean big sd is medium risk?
Take the overall mean of every mean and then compare how well the team matches that mean, lower is big risk, higher is smaller?
This might favour teams with lower b values overall, but if they're all lower then sd should be small. Mayhaps sd should be more important than mean?

Score_i = w_1 * sigma_i + w_2 * (1 - mu_i)?

Drought of available teams if impact is ignored could make this not as accurate, lack of available projects needs to be a priority, not just distribution

perhaps each team also has an inverse score based on the number of impact projects? like if team 3 only has 6, they have a score of 1/6, but team 9 has 13, they have a score of 1/13, and thats an additional multiplier

Create a score, that incorporates
- Preferences given?
- Mean of normalised b values
- Standard deviation of normalised b values
- Amount of possible projects

<u>Algorithm in terms of conflict resolution</u>
Given more than one good pairing, instead of just picking one, use an algorithm specifically to see which one would be best
What is best?
B value combined with available options?
- For team check projects in 3rd? Quarter

Need to run a bunch of code tests and experiments
Set up a graphing environment, sliders, scalers, normaliser
Scatter plot david was using

Defining fairness - at some point it has to be good enough, will always be trade offs, consider as many factors as possible

- Need to just put foot down and make decisions
    - Teams that put in preferences get allocated first
    - Rank all combinations somehow and say thats the order its important to go

Fairness
https://link.springer.com/chapter/10.1007/978-3-030-37157-9_1
- Need to choose between efficiency and fairness, whatever efficiency is in this case
- Perhaps during the allocation process would be worth keeping track of the fairness metric, like the MAD/SAD score for each team, or Gini Index
- Jain index used to track how fair the process was

https://psychologicabelgica.com/articles/10.5334/pb.684

https://www.anderson.ucla.edu/documents/areas/fac/policy/fox_ratner_lieb-8-16-04.pdf
- Given different pools of choices, different allocations will be made
- I.e. given looking at just one team and all options will likely be allocated something different than if given three teams and five options that overlap

https://pubsonline.informs.org/doi/10.1287/opre.1100.0865
- Price of fairness - making something more fair reduces efficiency

https://www.brookings.edu/articles/algorithmic-bias-detection-and-mitigation-best-practices-and-policies-to-reduce-consumer-harms/

Drought of options research
https://bioethics.hms.harvard.edu/journal/AI-resource-allocation
- Unbiased options provide most efficiency but truly are just unfair
-

https://www.mat.uc.pt/~lnv/talks/MOO-review-2020.pdf
- Pareto ordering - define a region of options that are all better than the outside?

Human in the loop allocation
https://link.springer.com/article/10.1007/s10458-021-09514-w
https://digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2017-189.pdf
https://arxiv.org/pdf/2301.07766
https://www.sciencedirect.com/science/article/abs/pii/S0951832020308292
https://journals.sagepub.com/doi/10.1177/20539517221115189

Enhanced stable matching
https://link.springer.com/article/10.1007/s10614-020-10006-4

https://www.mdpi.com/2073-8994/13/1/46
https://arxiv.org/pdf/2312.03006
https://arxiv.org/pdf/2306.11233

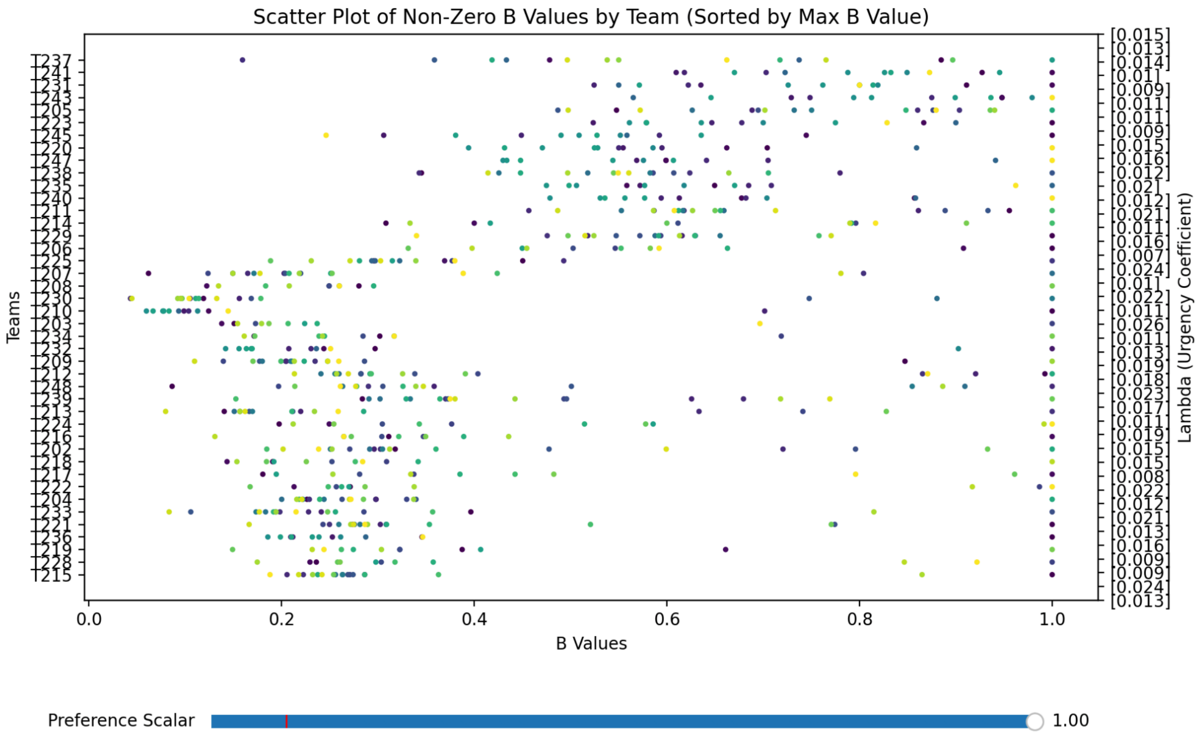
Comparing teams orders
https://towardsdatascience.com/how-to-compare-two-distributions-in-practice-8c676904a285
https://pubsonline.informs.org/doi/pdf/10.1287/ited.2013.0124
https://www.mdpi.com/2673-8392/3/1/6
https://rich-d-wilkinson.github.io/MATH3030/6-mds.html
https://www.tandfonline.com/doi/full/10.1080/1331677X.2015.1075139
https://uhra.herts.ac.uk/bitstream/handle/2299/12591/s153.pdf?sequence=2

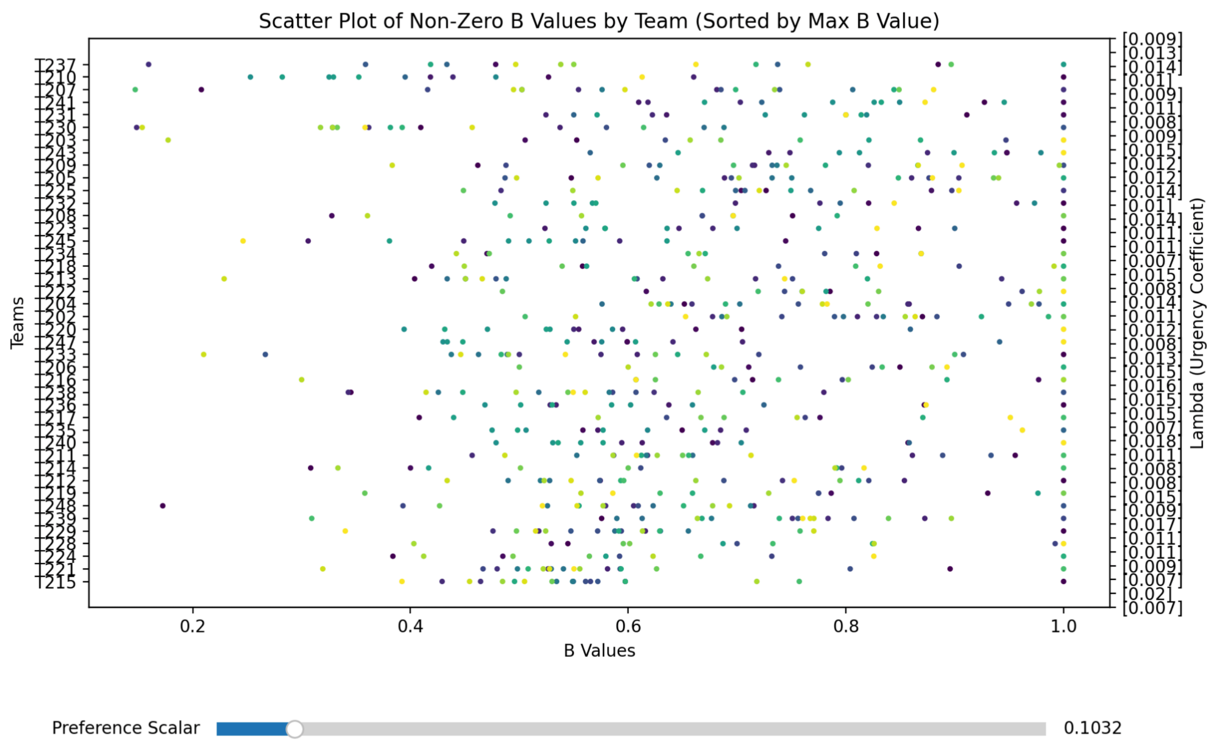
Properties
- Preference input - benefit
- Available projects - cost
- Mean - benefit
    - Normalised to what?
    - Maximum for that project? Same scale for mean between teams
    - Maximum for all projects? - same scale for standard deviation between teams
- Spread (standard deviation) - cost?
- How much allocating them a project would impact everyone else - cost?

Maybe E = benefit/cost? E = (goodness - impact on others)/remaining options? Should it be one static ranking or update after every allocation

Can’t rank by mean - won’t work. Doesn’t actually mean anything in terms of urgency or priority. More impactful would be a ranking of standard deviation (spread) and number of available projects

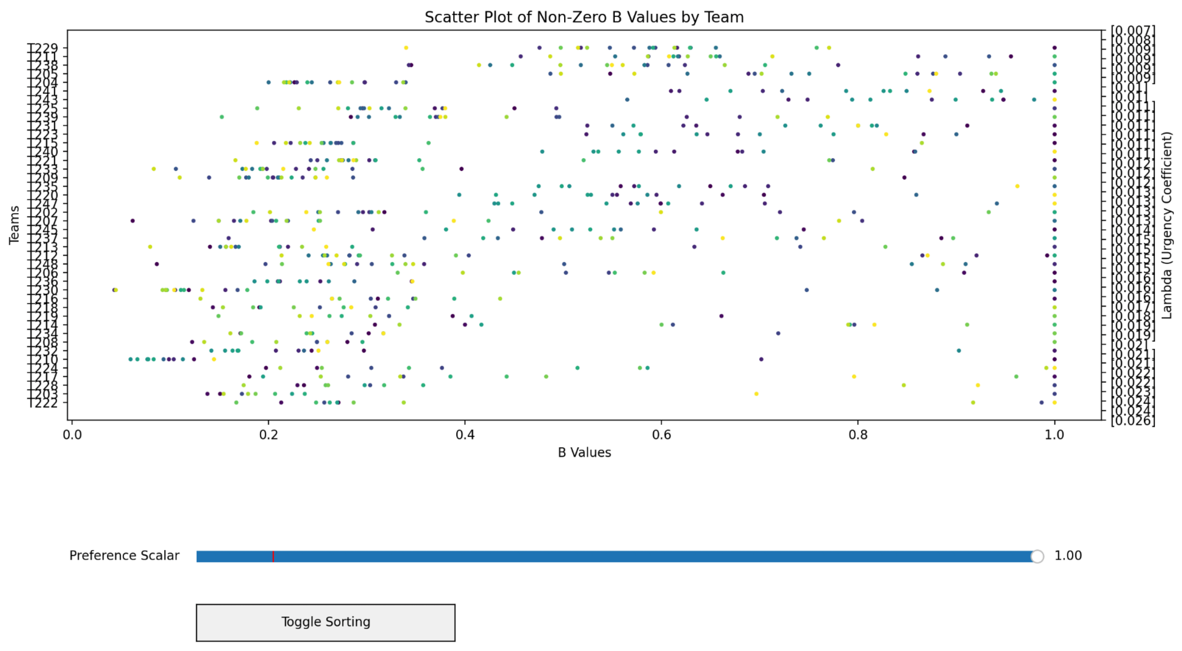
$\sigma = \frac{standard\ deviation}{number\ of\ projects}$, small is less urgent, big is more urgent





Ok so we need $\sigma = \frac{std}{num\ projects} + 0.5$ for preference, just to balance that to be more important.

Not 0.5 that would be b values, but maybe some scalar



Sorted by lowest to highest priority, can clearly see the bottom ones have big range and few options, top ones have better chances of getting a good one

Does Normalization Inflate the Spread for Less-Skilled Teams? BIG IMPORTANT POINT!!! I don't think it does, plus this is inflated b values

Got to remember we should be expecting 0.1 preference scale not 1, so it will normally be better, and then we weight the preference more.

Thats urgency defined. Now how about impact on other teams.

Each team has a set of available projects. Some of these projects will coincide with other teams projects.
Let team 1 have the set of projects $A = \{p_1, p_2, p_4, ..., p_m\} \subset \{p_1, p_2, p_3, p_4, ..., p_n\} = P$
Let team 2 have the set of projects $B = \{p_1, p_2, p_3, p_5, ..., p_n\} \subset P$
Count the fraction of projects in B that are in the top half of b values of A? Average the percentage across every other team. Gives each team a value λ as a percentage that their projects match the best of everyone else? Higher is worse, so its a benefit and they go first? Or its higher relative to number of teams to $\frac{\lambda}{number\ of\ teams}$ to make it more relative? And then

Need to reset thinking.
For each team, for every other team calculate how many of the first teams projects exist in the second teams top 50% of projects, as a fraction. Either average (bad because reduces importance of collisions with other singular teams over spread) or add up these values to get λ.

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,

$\lambda = \frac{\sum_{p \in P_i} N_j(p)}{|P_i|}$, higher lambda means more contested/priority?, lower lambda means less contested

Perhaps there would be potential to identify groups of high contested specific projects. i.e. i expect it to be likely that certain projects are contested by certain groups. By grouping them, it could be possible to go group by group to only have to worry about one or two collisions at a time? So a lambda for each team for every other team \lambda_ij, and group teams that have high lambda for each other?

$$\lambda_{ij} = \frac{\sum_{p \in P_i \cap P_j} \left(\frac{1}{\text{total projects}} \right)}{|P_i|}$$

This seems to link a lot with greedy allocation, will refer to that research and come back to this later. Move onto identifying low impact high value pairings

Time to maximise objective functions over 500 dimensions lets go!!!

Gotta find the least impacted projects that are good. I guess maybe just run through the normalised 1 values and check if they have a 'low' lambda. What is a low lambda?

Doesn't seem to be any distinction between unconnected lambda values and connected ones. Also, somewhere in the change from graph to graph2, i lost the three distinct groups somehow. Need to write my own code I think, not use chatgpt lol

How do we allow unfairness? At some point it has to be allowed. Easiest option is to do all preferenced teams first, or weight them more heavily. As discussed above can't go top down sorted by b value as thats not fair. Should determine whether more options goes first or last

Maybe a combination between sigma and lambda, i.e. measure of impact of not getting a choice combined with the range of choices

Can go all in on fairness min-maxing but that ends up getting an algorithm to do the whole allocation, which, again, is bad. **Need to give up on maximising fairness, and just improve it where you can**

Price of fairness - making something fair reduces efficiency
NOT UP TO THE ALGORITHM - ALGORITHM CANNOT MAKE THAT DECISION - ENTIRELY ON USERS SHOULDERS
Any algorithm implemented cannot make decisions. Can provide suggestions

Use Pareto to do a soft allocation, remove the bad options

- Team with many options that have low overlap, i.e. high lambda? Can ignore the bottom options

How can algorithms work in the system? Where are they involved

Can:

- Choose the order teams are allocated in
- Identify groupings of teams
- Identify overlap in projects
- Choose which projects to display for each team
- Choose how to show teams

Can't do things that would influence choice

- Selectively showing data
- Abusing visualisation to influence choice

Can only use algorithms on objective actions

- These are the projects for this team
- These are the teams the team overlaps with
- This team will be harder to allocate for if it goes last i.e. should be done earlier

These are objective notions in the dataset, as unsubjective at all

Take the bias from the user in subjection situations by defining an objective method

- Subjective choice in order into deterministic order

Do not interfere with the human in the loop

- Show all the data, allow the human to make the decision

In situations where efficiency improvement does not reduce fairness, use an algorithm. In situations where fairness would be reduced, do not use an algorithm.

Perhaps it would be better to say where something is already unfair, and using an algorithm would increase efficiency, then use an algorithm. If it is fair, then don't.

Efficiency cost is not a zero sum game

Could a graph colouring work? Actually probably not because its not about areas or overlaps and its more about weighting for how good a pairing is.

https://shubhamjain0594.github.io/post/tlds-arvind-fairness-definitions/#individual-fairness