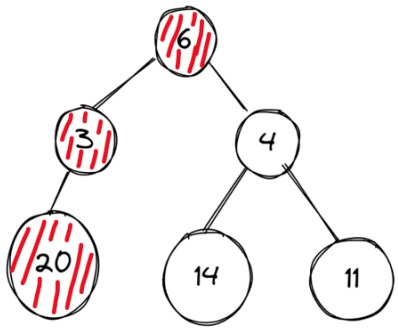
Minimization of Greedy Allocation Notes

Greedy algorithms make locally optimal choices at each step with the local information as per predefined criterion, with disregard to future consequences or fairness.



- Greedy algorithms may not always find the best possible solution.
- The order in which the elements are considered can significantly impact the outcome.
- Greedy algorithms focus on local optimizations and may miss better solutions that require considering a broader context.
- Extremely time efficient as it only considers local optima at each step as opposed to global solution
- Simple to implement as the logic is you need to find local optima
- Very useful in approximations when global solution is hard to find

https://www.geeksforgeeks.org/greedy-algorithms/#what-is-greedy-algorithm

https://library.fiveable.me/combinatorial-optimization/unit-8/greedy-approximation-algorithms/study-guide/LGB3wBj4i0e6Q8Xt

ILP algorithm is not greedy (it finds global solution), but differences in b-values can cause the algorithm to immediately prioritize same high teams to projects, thus even though it is not greedy algorithm, it can be skewed into exhibiting greedy behaviour. So, the reduction in the effect large b-values have on the system is paramount to reduce greedy allocation.

https://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

Methods to reduce this

Normalization with the equation $normb(T,P) = \frac{benefit(T,P)}{maxbenefit(T)}$

With T being team and P being project, will scale each teams benefits according to their maximum potential, a scale of 0 to 1. Thus, the teams will then be ranked for projects purely from their potential (not potential to complete it well, but just simply their maximum potential project). This will completely remove the effect of high absolute b-values.

https://www.mit.edu/~dbertsim/papers/Fairness/The%20Price%20of%20Fairness.pdf

Will result in a much less optimal system, namely because weak teams may steal away stronger projects if their normalized benefit is higher, even though they may be extremely suboptimal, perhaps detrimental allocations. A weak team could have high benefit scores of 1, 0.9, 0.8 towards projects when in reality they are not good fits for any of them, i.e. they are the best projects relative to the teams maximum potential, but the team is not the best for the project.

Two ways to reduce the negative effects of this normalization:

- Thresholds: by implementing certain thresholds (tiers) that teams need to be even considered for projects, means that the highest value projects are no longer at risk of being taken away by suboptimal teams (as these teams are in a lower tier), however the normalization

is still applied so that teams within the tier are evaluated against each other based on their maximum potential.

- Weighted normalization:
- $weightednormb(T,P) = a * \frac{benefit(T,P)}{maxbenefit(T)} + (1-a) * benefit(T,P)$
- Introduces a second term, with first being normalized scores and second being absolute scores. Now that both are being taking into account, the scaling factor a can then be tweaked in order to determine how strongly normalization, or the absolute benefit should determine the weighted norm.

Kind of similar to the weighted normalization is implementing upper confidence bound:

$$UCB_{ij} = b(T_i, P_j) + \sqrt{\frac{2\ln(T)}{n_{ij}}}$$

$b(T_i, P_j)$ term (exploitation) will ensure absolute benefit value is accounted for, however the $\sqrt{\frac{2\ln(T)}{n_{ij}}}$ term (exploration) will consider teams that have not received many allocations, so this could be used in a system were the allocation is repeated. This dynamically flip flops between prioritizing the high b values of optimal teams and teams that are not considered optimal for many projects.

https://www.cs.princeton.edu/courses/archive/fall16/cos402/lectures/402-lec22.pdf

Greedy Allocation

https://www.cis.temple.edu/~jiewu/teaching/Spring2021/Chap4.pdf

https://www.geeksforgeeks.org/greedy-algorithms/

https://brilliant.org/wiki/greedy-algorithm/

https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/lectures/13/Small13.pdf

UCB

https://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/

Current ILP aims to maximize total score, so we could potentially minimize the total score. Instead of greedy allocation causing the same teams to always be prioritized to same projects, could attempt to iterate through the allocations, and try to minimize the largest variation in score (i.e. Try to get rid of the absolute worst allocations).

This is similar to bottleneck minimization, where instead of minimizing or maximizing scores, it aims to reduce the bottleneck of the allocations. A bottleneck in this context is the lowest b-value of any team to project allocation i.e. the absolute worst team-project pairing.

Tabu search could be added to this. Iterations are made to the allocations to make small variances of the allocations, in this case by changing slightly changing allocations, and storing these to memory in order to keep track of all the changes that have been made. Algorithm will keep exploring other possibilities without retrying those already added to the list. By avoiding allocations that have already been made, the effects of local optima skewing the algorithm into behaving greedily can be avoided.

https://leeds-faculty.colorado.edu/glover/fred%20pubs/376%20-%20TS%20-%20Principles%20of%20Tabu%20Search%20-%20Glover,%20Laguna%20and%20Marti.pdf

BFA defines a baseline share of the total benefit score available across all projects, to ensure that all teams receive their fair share. It can be defined mathematically by the equation:

$$BFA(T) = \frac{1}{n} \times Total\ Benefit$$

Where T is an individual team, and n is the number of teams.

All teams are guaranteed a certain proportion of the total score, meaning no teams are disproportionately advantaged or disadvantaged by an allocation.

Pareto efficiency is a concept brought from economics that could be applied to this allocation algorithm. It is the concept of a system in which resources cannot be allocated to make one individual better off at the expense of another becoming worse off.

https://www.investopedia.com/terms/p/pareto-efficiency.asp

So, we would in this project, be attempting to find the set of allocations in which you are unable to re-allocate any project as it would decrease another teams benefit score.

- Algorithm should be able to look past local optima and understand that improvements for some teams may come at the cost of many other teams.
- Doesn't necessarily result in global optimality of the solution, as it focuses more on locally changing allocations to reach a pareto-optimal state, however this does not mean that the system has optimally allocated teams in terms of achieving a high total score, i.e. allocations may not be the strongest.
- Doesn't guarantee fairness, as high ranking teams may still be assigned high ranking projects, and changing them may hurt other teams.
- Maybe combined with fairness constraints so to balance the optimality of solution from pareto but also try to make sure that the system isn't cut-throat. Fairness constraints could include:
- $B(T_a) \geq \frac{Total\ Benefit}{n} * a$
- $BFA(T) = \frac{1}{n} \times Total\ Benefit$