Construction Heuristic

The construction heuristic uses a task-difficulty prioritization approach with multi-pass refinement. Initially, we sort tasks by their "eligible count" (the number of workers qualified and available for each task), processing harder-to-assign tasks first.

The random initialisation first considers eligible worker count and average cost. It prioritises tasks that have few eligible workers and cost the lowest on average. It then performs 5 re-sort to assign any remaining unassigned tasks, and performs 3 swaps to reduce overall costs. Initial random initialisation functions did not give my problem a good starting point. That made me decided to include more conditions to allow my starting point to be, thereby narrowing the search for a better optimum.

Destroy Methods

The final destroy operator used was the default option of removing a percentage of assigned tasks at random. Initial remove_frac value given was 20%. However, because my initial starting point was already better after changes were made, lowering the value of remove_frac to 15% gave a more optimal solution.

Repair Methods

Repair operator used takes inspiration from the way the problem first constructs its initial solution. It takes all unassigned tasks from the destroy method and sort them according to which tasks has the least amount of eligible workers. For each eligible worker, it calculated the increase in cost when the task is assigned to these workers and assign it to the worker that contributes to the overall costs the least. This is similar to the original repair provided but considers the difficulty of assigning as well as the costs to ensure that it always minimises costs.

Weights Adjustment Strategy

Our ALNS implementation uses a reward-based weight adjustment strategy with the following parameters:

- Weight Vector [40, 20, 5, 0]: Representing rewards for finding a new global best solution, a better solution than current, a worse but accepted solution, and a rejected solution, respectively.
- Decay Parameter ($\lambda = 0.9$): Controls how quickly the algorithm forgets past performance, with higher values giving more weight to recent results.

This adaptive weighting mechanism ensures that successful operators are selected more frequently while maintaining exploration by occasionally selecting less successful operators. The relatively high decay parameter (0.9) allows the algorithm to adapt quickly to changing effectiveness of different operators throughout the search process.