Formulation & Algorithm Implementation Notes`

The accompanying problem formulation pdf file includes the specific formulation I implemented.

Primal/Dual: These use the Gurobi library, which takes LPs in an Ax = b matrix form. I use numpy to easily translate the problem formulation constraints into a format Gurobi understands using matrix conversion helper functions.

Online assignment: You will notice that the online algorithm takes betas and the primal allocations as an argument. I include these as a debugging measure so that it is easy to calculate the online comparison term, but they are not necessary for the implementation.

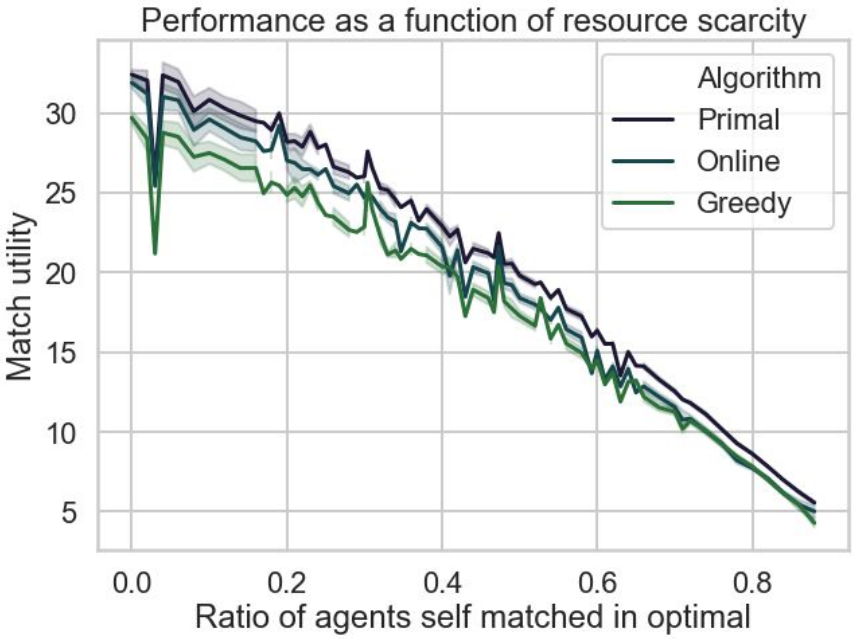
Greedy algorithm: I implemented this such that for each time round, for each non-matched agent, it matches with the maximum weighted pair until no pairs remain. There are different ways to go about this which may be more valid, including solving a small LP for that time step.

Performance Comparison of Matching Algorithms

The python notebook includes code to run the experiments I discuss below.

Experiment one compares the offline primal, online dual-based, and online greedy matching algorithms across various resource scarcity scenarios. In this experiment, match quality weights between each resource and agent are drawn randomly from a uniform distribution and remain constant over time. Resource scarcity, as measured by the ratio of agents that remain un-matched in the optimal primal solution, is manipulated by varying the number of copies and use times of each resource.

The graph below shows that online dual-based assignment out performs greedy assignment, especially when resources are more abundant. As resources become more scarce, the difference becomes less pronounced. The offline primal solution remains superior to both online approaches across scarcity scenarios.



J = 50, d = 4, I = 2

This scenario is not necessarily a realistic model of homelessness. In fact, online greedy assignment may perform even worse in several common homelessness matching scenarios where match weights are not random. Suppose the three following situations, each of which consists of two types of homeless agents (type-A and type-B), and two types of resources (type-X and type-Y).

Scenario 1

Here, resource X is abundant, but provides a small and consistent utility to both Type-A and Type-B agents. Resource Y is much less abundant, and provides a utility that varies depending on the agent. Resource Y benefits agents of type-B significantly, and type-A agents much less so. Regardless of the agent, a type-Y resource is preferable to one of Type-X.

	Agent Type-A	**Agent Type-B**
Resource Type-X (5 copies, 15 round use)	.05	.05
Resource Type-Y (10 copies, 5 round use)	.1	.9

Scenario 2

Here, resource X and Y are equally abundant. Resource X provides a fixed high utility to both agent types, while resource Y provides a lower utility that depends on the matched agent.

	Agent Type-A	**Agent Type-B**
Resource Type-X (5 copies, 3 round use)	.99	.99
Resource Type-Y (5 copies, 3 round use)	.95	.05

Scenario 3

Here, resource X and Y are both very scarce, with only one copy that will be allocated permanently. The utility distribution of matching remains similar to scenario 1.

	Agent Type-A	**Agent Type-B**
Resource Type-X (1 copies, T round use)	.05	.05
Resource Type-Y (1 copies, T round use)	.1	.9

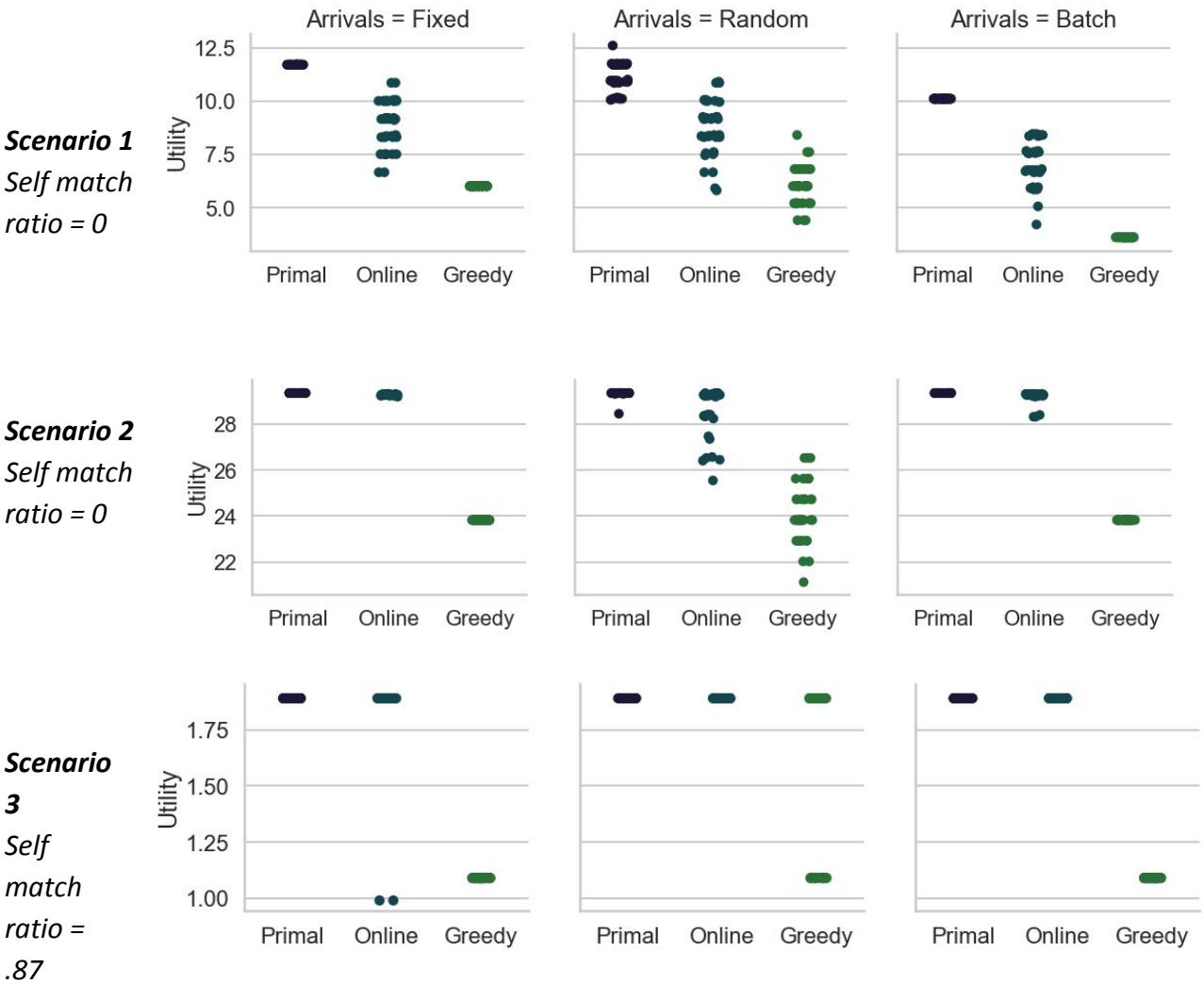
Arrival Models

Another factor to consider is the arrival model of incoming agents. The distribution of Type-A & B agent arrivals will affect resource uses differently under different matching policies. Three simple arrival models to consider are:

Alternating arrivals: Type A & B agents arrive in an orderly, sequential manner. A Type A agent will arrive, followed by a Type B agent, which will continue to alternate. This is not that interesting, and mostly becomes useful for the alpha

Stochastic arrivals: Type A & B agents arrive in a random order, and alternating types are not guaranteed.

Adversarial arrivals: All low utility agents arrive, followed by all high utility agents. This is still a lot of room in terms of other adversarial arrival scenarios / weight distributions to test.
A second experiment compares utilities under each of the above scenarios and arrival models–each setting was simulated 40 times with the allocation utility distributions plotted below.

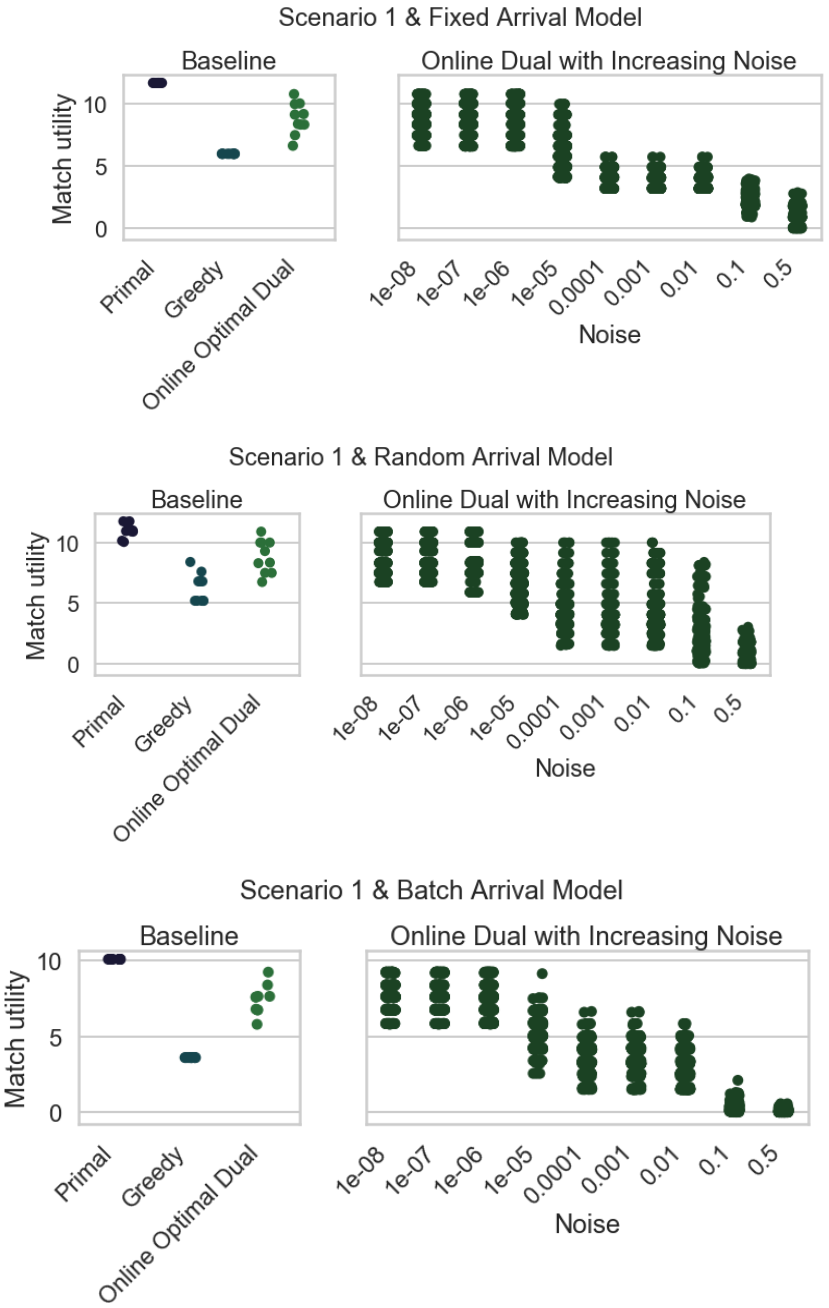


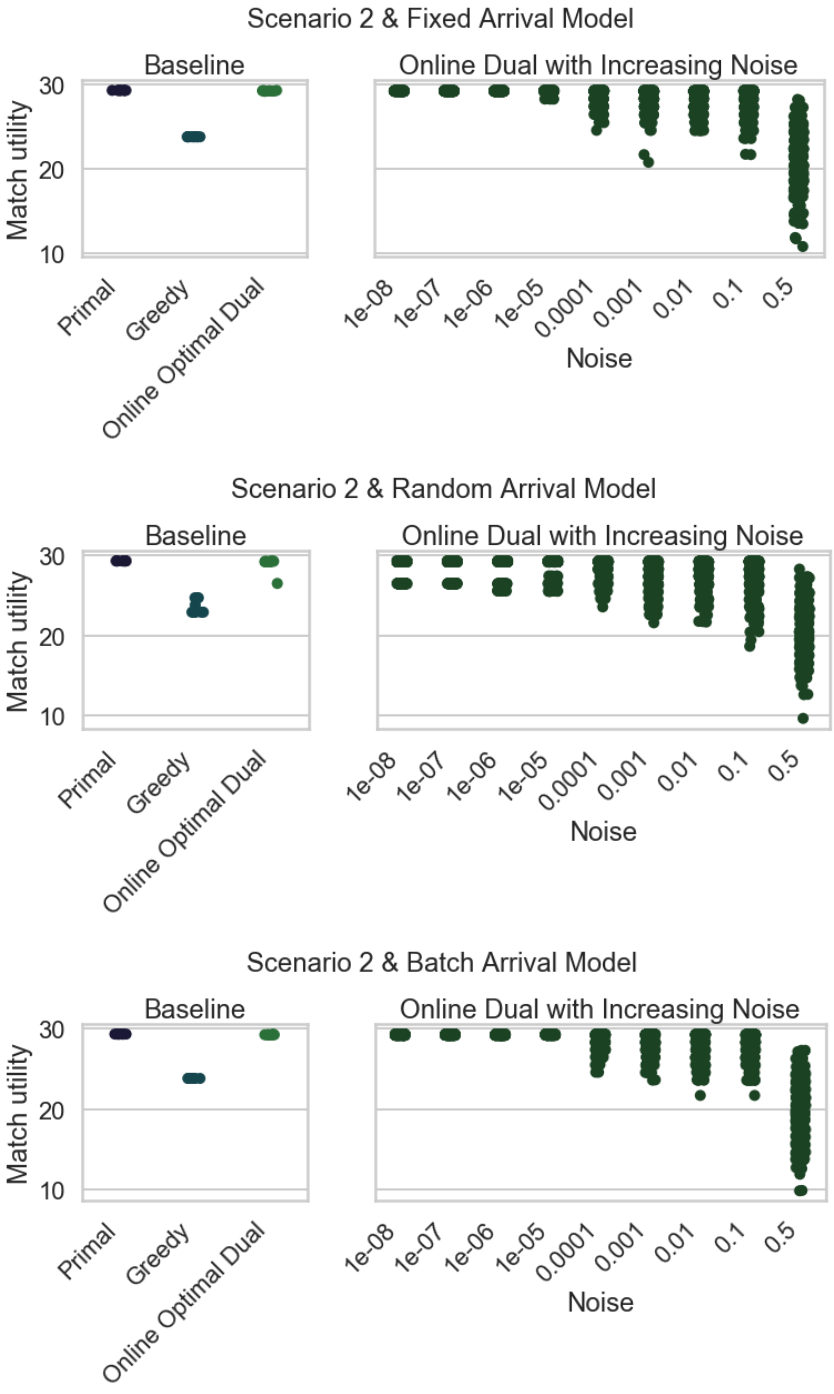
J = 30, d = 4, 40 runs at each configuration

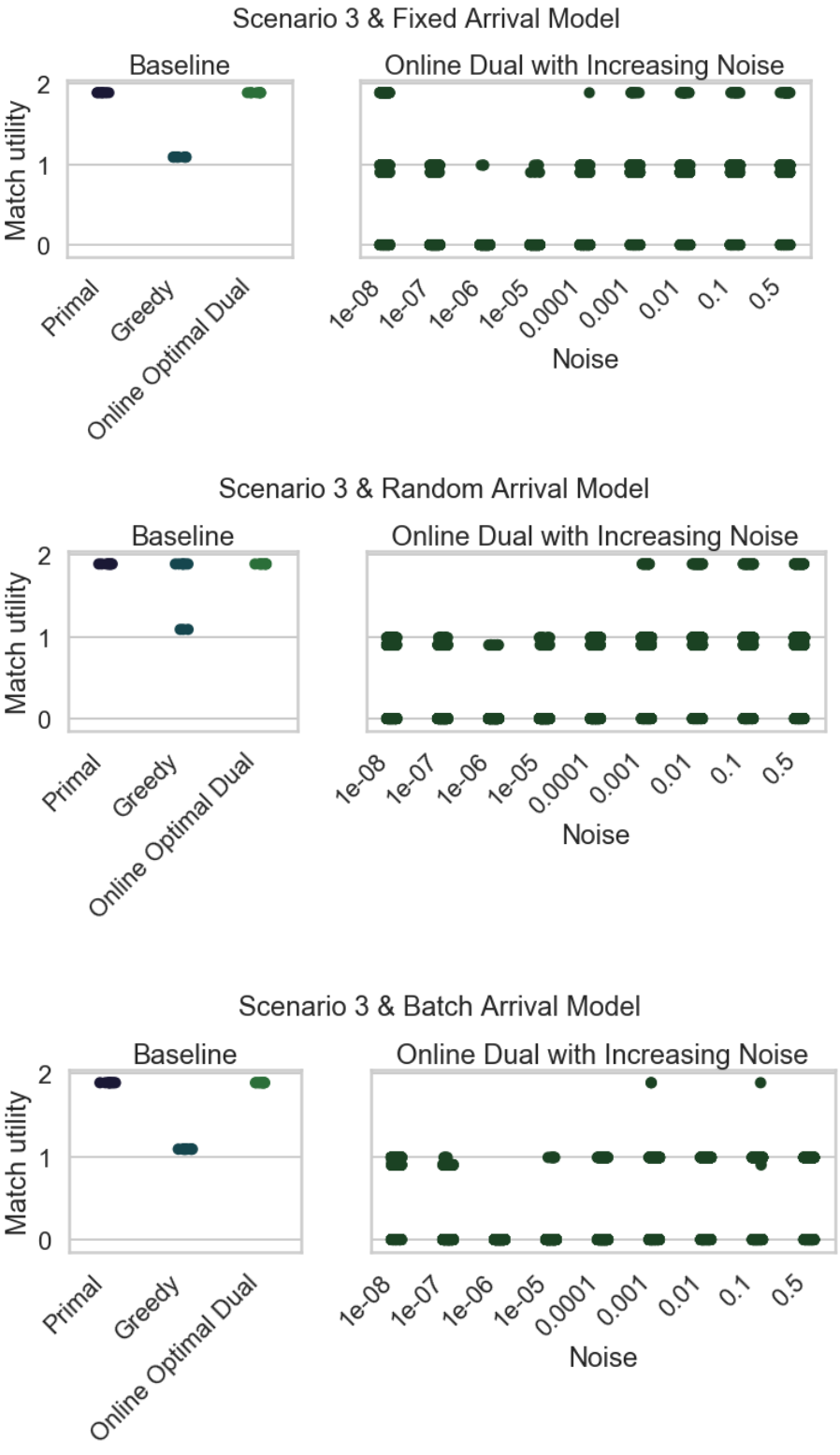
Allocating with non-optimal dual variables

In practice, we don't have knowledge of future arrivals, and therefore can't know the optimal dual variables to use in online dual-based allocation. By adding noise to the dual

variables before allocation, we can get a sense as to how robust online dual-based allocation is to noise in the alpha dual variables. The below graphs show how performance deteriorates as noise increases in each of the scenario / arrival model combinations. For each simulation, the baseline models are run 10 times, and for each of these 10 baselines, 40 randomly generated noise vectors are added to the alpha duals.




Scenario 2 & Fixed Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
30
20
10
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise
Scenario 2 & Random Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
30
20
10
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise
Scenario 2 & Batch Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
30
20
10
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise


Scenario 3 & Fixed Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
2
1
0
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise
Scenario 3 & Random Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
2
1
0
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise
Scenario 3 & Batch Arrival Model
Baseline
Online Dual with Increasing Noise
Match utility
2
1
0
Primal
Greedy
Online Optimal Dual
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
0.5
Noise

Where noise is scaled as a percentage of the weight values. These show that (a) online assignment is sensitive to noise, and (b) sensitivity depends on the arrival scenario. These results may be somewhat pessimistic depending on the dual estimation method. Since alphas are sparse in practice and follow a distribution somewhat similar to copies becoming available, if the noise is better-behaved, results may be better.

Allocating Based on Simulated Alphas

A simple approach to estimate optimal alpha variables is to simulate a scenario multiple times and average the optimal alphas. Since resource allocations will follow roughly a periodic pattern, this may provide an informative approximation for the online assignment algorithm.

Specifically, this process involves:

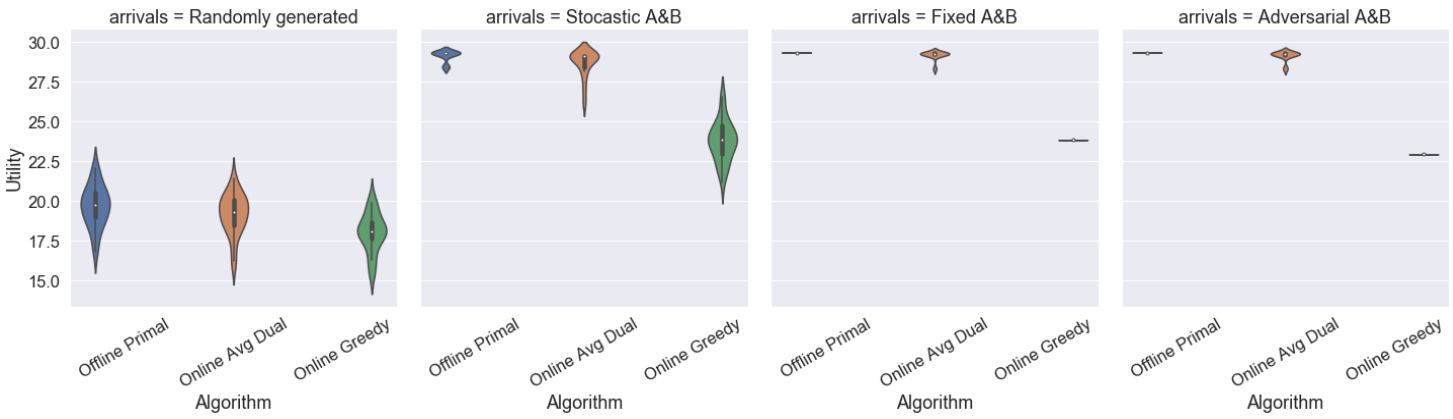
1. For N_SIMULATIONS:
    a. Generate pairing weights according to the arrival model, resource copies, and use times. Keep arrivals, copies, use times consistent across simulations.
    b. Run the solvers and save the optimal alpha variables

2. Construct an average alpha dual vector based on the mean of the simulated runs.

3. Run N_TEST simulations using the average alpha dual vector to make allocations in the online dual algorithm. Plot variability to get a feel for how well this works on average.

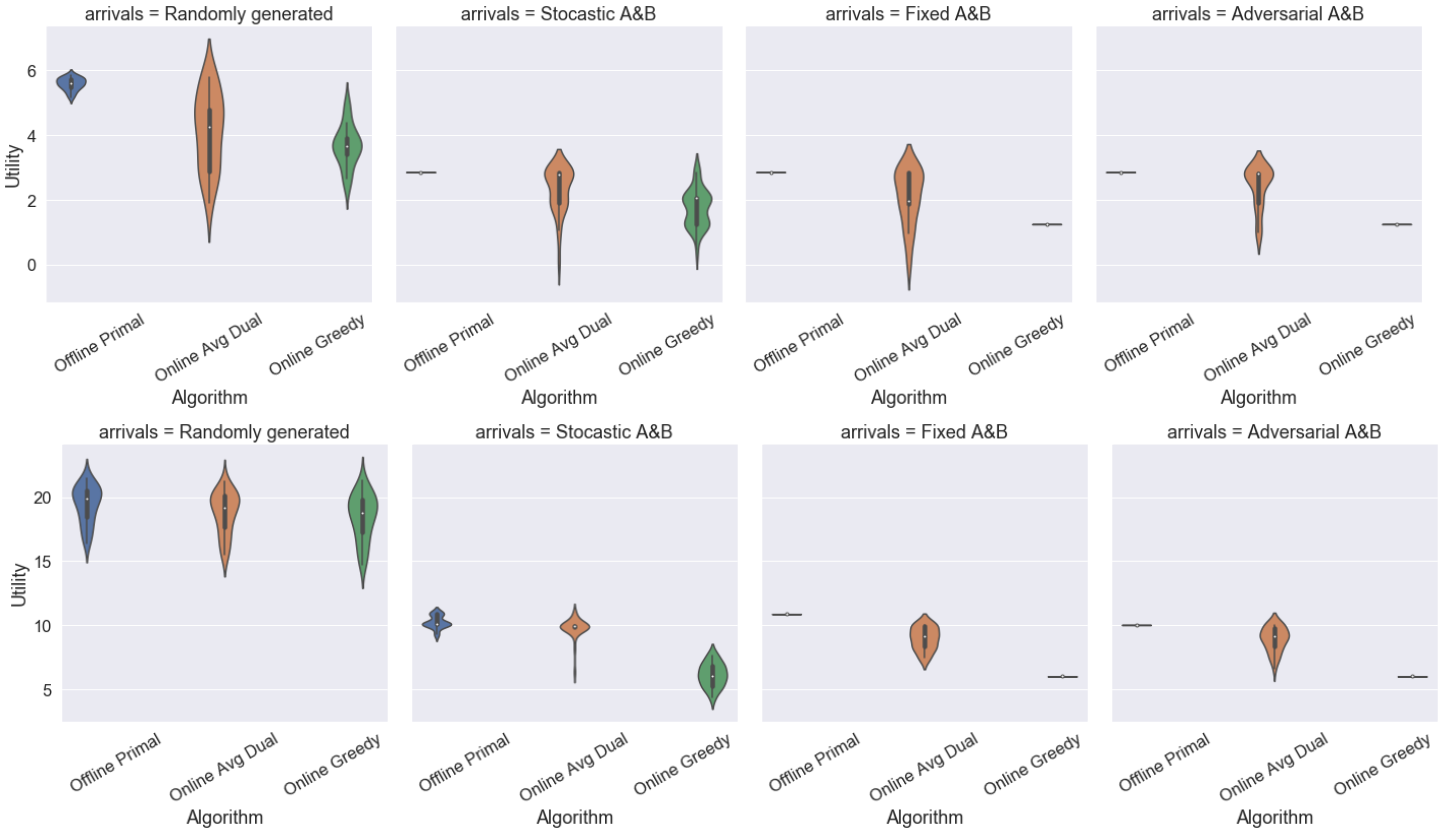
For the scenarios above, this works surprisingly well (all plots N_SIMULATIONS=50, N_TEST=30, J = 30, I = 3, d = 2):

Scenario 1:



Scenario 2:

Scenario 3 (with 3 copies each instead of 1):



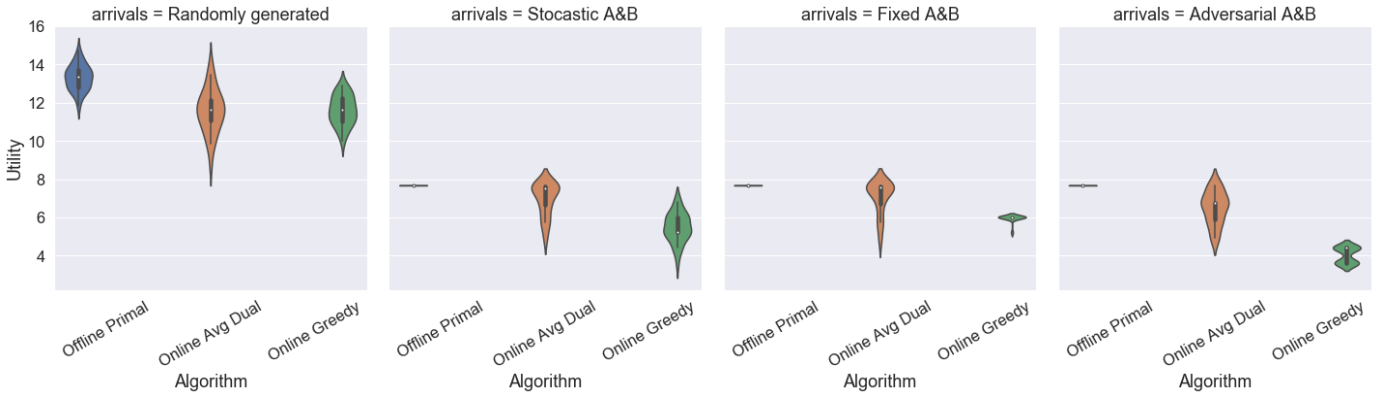
How robust is this?

Keeping the scenario 1 weight distribution, change the agent wait time, resource copies, and resource use times.

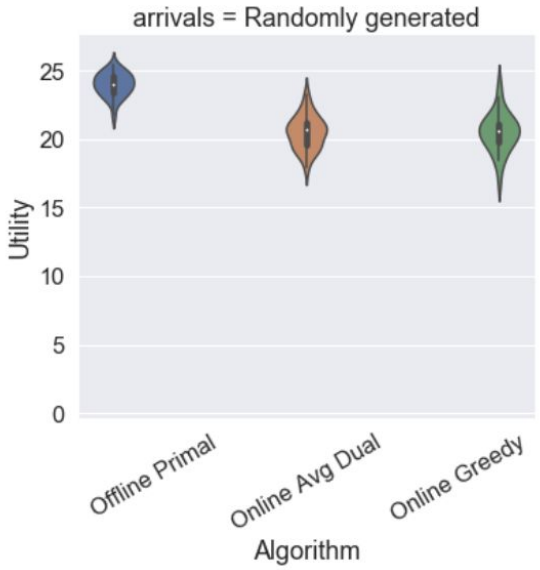
k = np.array([1,13,16])

c = np.array([J,3,4])

d = 6



Increase the number of resources and run on randomly generated weights. We would expect an average alpha vector to not be informative in this context:

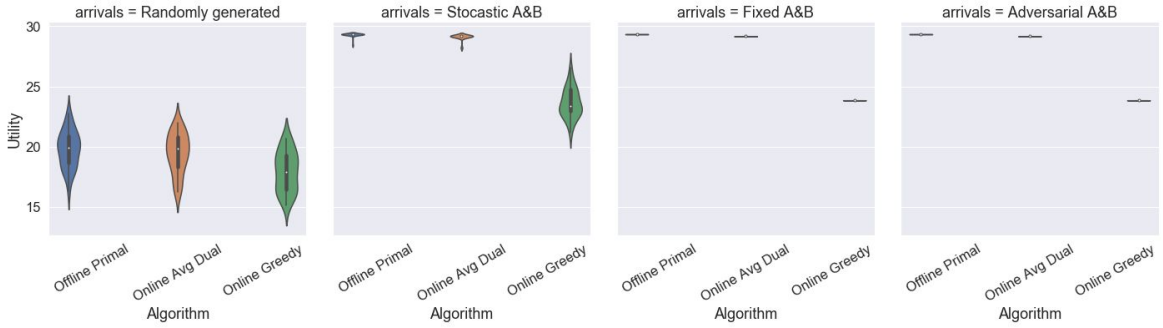


Estimating alphas based on the average of several runs has the drawback in that it requires scenarios of equal size between the 'training' and 'testing' phases. However, due to the periodic nature of the alpha variables, this could be addressed by either:

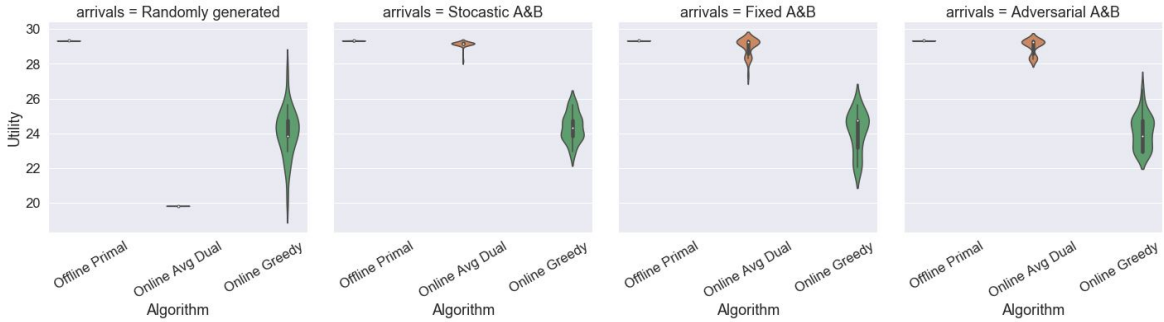
1. Averaging scenarios with a larger J and truncating to the test size, or
2. Extending the periodic trend observed in a smaller J to a larger scenario

I also tried averaging based on arrivals in one model, then applying this mean vector to other situations

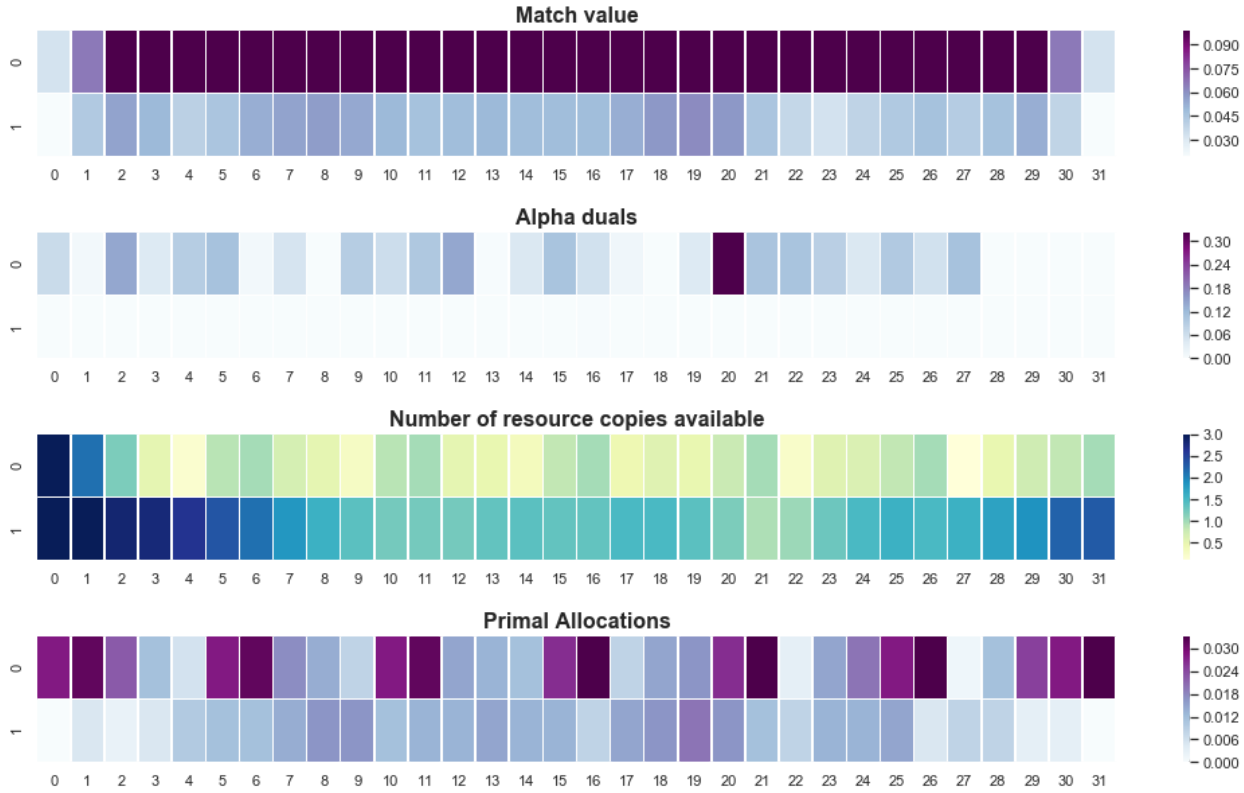
Train on stochastic, test on all other models



Train on all, test on stochastic



I found the good performance of online assignment based on averaged duals of stochastic arrivals very surprising. When we look at the online assignment term, however, it makes some sense. The - sum alphas - beta term is basically forcing a penalty when dual variables are high. When alpha duals peak around when resources become available, it is forcing online assignment to wait for a better match candidate as opposed to matching immediately. However, the relationship between the number of available copies and duals isn't always clear. For example, I have a hard time definitely saying the relationship based on 20 simulated stochastic arrivals:



Looking into this relationship further is still very interesting. Even in scenarios where weights are distributed more randomly, there is probably an optimal way to be more or less greedy based on the abundance of other resource types and the likelihood of encountering a better match in the near future.

If I were to continue working on the project, things I would do next are:

1. Finish the simulation forward method based on the primal and average dual techniques we discussed.
2. Return to the problem formulation to see whether there are solutions to the tie issue.
3. Manually engineer dual alpha vectors that I think should work well in different contexts to better understand how the dual is impacting online dual assignment.