4.4 - Discussion of what would be necessary to achieve the target levels

100% Coverage of my stated functional requirements from LO1

Despite this criteria being achieved on paper, there is a lot of room for improvement. 100% test coverage isn't just "have at least one test for each thing", but rather "Can we be almost certain that these tests are trustworthy".

To fulfil this criteria to a higher standard, I would test all functional requirements to a higher depth, employing different testing strategies where possible. Weighting the random points closer to no-fly zones and creating some difficult tests by hand could give a higher confidence in the system overall, getting much closer to achieving a more realistic 100% functional requirement coverage goal.

High repetition targets for random tests (>50 in CI)

The low performance of the delivery path calculation hinders the ability of the pipeline to run in a reasonable amount of time with higher repetition counts. To increase the feasible repetition count for the pipeline, either the performance of the delivery path calculation needs improved for larger delivery counts, or tests need to be performed on only a couple of deliveries - which may impair the effectiveness of the testing.

Once the performance is improved, it would be realistic to see 50-100 repetitions of the random tests within the CI pipeline, combined with the weighted random points from above, this would give me a much higher confidence that any given push that passes the pipeline really does meet the functional requirements.

Higher confidence in my non-functional performance (tests on larger amounts of deliveries, 10-15+)

As with the above criteria, the most pressing issue holding this back is the performance of the delivery path calculation.

Increasing the performance of the system for larger delivery sizes would likely involve dropping the permutation approach of estimating and then picking the best ordering of deliveries. Instead, it would rely on a less optimal but much faster heuristic based approach that still makes all deliveries correctly.

Once an algorithm is in place that doesn't grow factorially[1], running tests with a higher amount of deliveries on the pipeline should be a breeze, and with a more sensible (yet still not great) exponential growth, 20 deliveries would take a little under 10 seconds, which is much more suitable for the pipeline, increasing the confidence we could have in the system.

Negative tests for stated functional requirements (tests that should intentionally fail - negative coverage essentially)

For each functional requirement, I would identify cases where a failure state should be entered at some point. For example, if AStar was given a point that was too far away or in a no-fly-zone, it should eventually hit some path cost limit and return an empty path.

These kinds of scenarios aren't currently explicitly tested for, and are areas in which we don't know how the implementation would react, representing a further lack of complete coverage of the system.

Negative tests are just as integral as positive ones, and so eventually for a production system, every functional component should have as many negative scenarios coverage as are identified, even if they're hand written and not generated ones.

---

1. This level of performance degradation is *quite* bad. Given it takes 30 seconds to do 10 deliveries, calculating the optimal route for 20 would take ~750,000 years on my machine... not exactly appropriate for the pipeline ↩