R1, R3: The system should be able to find the shortest path by avoiding no-fly zones/The system should be able to classify orders based on their attributes.

|  |  |  |
| --- | --- | --- |
| Class name | Target Coverage | Actual coverage (Calculated by IntelliJ) |
| LngLat | 100% | 83% |
| Polygon | 100% |  |
| Card | 100% | 96% |
| OrderChecker | 100% | 87% |

The adequacy criteria for R1.1 (Coverage of LngLat and Polygon) is sufficient, as all lines in the methods which are relevant are executed at least once (excluding toString, hashcode, etc). The defect density is also low, only occurring in the case where three vertices are collinear, which produces an incorrect visibility graph.

However, the same cannot be said for R1.2. This relies on a third-party library JGraphT and a random number generator to produce restaurants.

* Limited no-fly zones
* Random generator could miss a class of coordinates which are more prone to errors

One way that some errors might not be caught for R3 is if in the real-world context where these card details need to be authenticated, is if the card does not exist. Therefore, we would need a way to check with the relevant card provider and check that it is indeed an existing card capable of transactions.

* Using category partition, I have extensively tested every combination possible, which ensures that this requirement is met.

R2: The system should take at most 60 seconds to find the shortest paths for all orders in each day.

|  |  |  |
| --- | --- | --- |
| System name | Target performance level | Actual performance level |
| Order validation |  |  |
| Visibility graph generation | 3ms (10-20 restaurants) |  |
| A-Star search |  |  |
| Entire system | 0.4 seconds | 1.4 seconds (First run)  0.6 seconds (Subsequent) |

By simplifying moves to nodes, there will be less comparisons between objects and thus the tests will not suffer from floating point errors. The performance criterion for R2 is adequate for the current dataset and is the only way that we can test this requirement directly. However, since I have no reliable process for generating more complicated data (Bigger orders, longer paths to restaurants), there could be certain configurations which do not have a passable execution time and thus fail to meet this requirement.

In the performance adequacy criterion that I used, we can see that the entire system takes 1.4 seconds to run for the first time, decreasing to around 0.6 for subsequent runs. This is partly due to IntelliJ’s caching of data received from the REST server, which take up the bulk of the run-time enabling the next executions to have a drastically decreased performance level.

How to achieve target levels:

R1, R2:

The target coverage levels have been met, although they are not at 100%, the missed methods are *toString* and *hashcode*, which we do not need to test. However, to get a stricter guarantee that the requirement is met, there could be a custom no-fly zone generator instead of using ones that have already been provided in the REST server. This generator would produce different types of no-fly zones – convex, concave, as well as other areas which have an irregular shape. However, due to resource constraints, it was not possible to develop a custom generator in the timeframe of one semester with just one person.

R3:

To achieve the target levels, I would need to be able to carry out further performance tests on the efficiency of my system to ensure that it performs as close as it can to the target level. For example, there could be more variety in the orders, such as having more restaurants and pizzas, expanding the types of cards I accept, and increasing the number of orders per day. By doing this, I could get an idea of the limit of my system and try to improve its performance in the subsystems where it took the longest to execute.

Furthermore, I would also need to improve the run-time of my visibility graph generation. Currently, it generates the graph with O(n3), where n is the number of nodes (Restaurants, vertices of no-fly zones, etc). However, there are some advanced algorithms that complete in O(n2logn), or even in some cases, O(n). For this project, these were too advanced for me to be able to both understand and implement within the given time frame. However, in a professional workplace with proper resources, both in terms of time and personnel, this would be an achievable goal which would allow this system to produce better performance levels and reach the target that I have set.