Dynamic Distribution through the city of Amsterdam
Computer Science bachelor project 2018

Ilja Bakx, Jelle Eysbach, Chris Mostert, Casper Schröder

Supervised by:
Stijn van Schooten
Matthijs Spaan
Johan Los
Otto Visser
Huijuan Wang

July 4, 2018

Acknowledgements

We would like to express our very great appreciation to Matthijs Spaan, Associate Professor at the Algorithmics Group of the Faculty of Engineering, Mathematics and Computer Science, and Johan Los, PhD at the Department of Maritime and Transport Technology, for supervising this project as representatives of the Delft University of Technology. We would also like to offer our special thanks to Stijn van Schooten from MakeTek for guiding our project.

Executive Summary

This report explains the design choices, implementation and results of a software engineering project commissioned by MakeTek. The team was tasked with making a system that could solve bike delivery scheduling problems with limited bike carrying capacity, time windows, dynamic delivery additions, movable pickup points and delivery time estimates.

The project started with two weeks of research, which concluded that the main algorithm should be a Genetic Algorithm (GA). The final product contains an app for visualising results as well as a backend that runs the genetic algorithm. Both systems are written in TypeScript and built upon a boilerplate provided by MakeTek. To calculate routes an open source routing provider is used, and the project is executed with an agile development workflow to ensure the product conforms to the client's expectations.

The implemented system contains almost all desired features and the missing ones were chosen to not be included due to time constraints and limited functional benefits. It is well-tested, extensible and adheres to software quality standards.

A solution is constructed by first clustering deliveries by location to distribute them over the bikes, then the genetic algorithm is run on each cluster separately. At the end of every generation of the GA the intermediate best solution is saved in a database. This ensures a valid solution is available at all times. Finally, postprocessing can be run on the final solution which checks if it is more efficient for a deliverer to wait between certain deliverers to prevent them from being early.

Testing shows that the implemented system and its features work as intended on different datasets. The effect of different parameters on the performance of the genetic algorithm is explored, with the following conclusions:

- The algorithm should have enough opportunity to explore the solution space. This is achieved by setting an appropriate mutation parameter.
- No significant performance differences are present between single point, two-point and uniform crossover.
- Tournament selection converges faster than roulette selection, but explores less of the solution space.

The resulting system successfully implements all of the requirements as set by the client. It includes an algorithm that converges to an optimum, it can adapt to new changes and a solution can be requested at any time during runtime. It can be concluded that the project is brought to a successful end.

Contents

1 Introduction

According to a recent study by Rabobank, flower and potted plant expenditures are expected to increase by 2% per year in Europe and North America. With millennials being likely to buy more plants online, the sales are expected to keep growing over the coming years. [21]

Thus, when running an online flower shop in the capital of the Netherlands, it is important to be able to keep up with the growing demand. To be able to do this, it's important to make the be able to efficiently plan a route so that all customers can be served on time, and make this planning scalable.

This is where this project comes in. The goal of the project is to design a system that a flower shop can use to schedule deliveries of online flower orders, with the customer being able to specify a time window for the delivery.

The end goal of the project is to create an open platform where initially only Amsterdam based companies can work together to combine deliveries and reduce their time on the road. Consequently they reduce their impact on traffic density and pollution, and increase overall road safety within the city.

Solving this problem will enable companies using this platform to supply additional options, tighter time windows and punctual same day delivery. The companies can use their deliverers more efficiently, and the deliverers themselves will be able to do their jobs more efficiently by following the route given by the platform instead of making it themselves.

The project is supplied by MakeTek, a small technology start-up created by students of TU Delft [6]. They coached the team for 11 weeks, where the first two weeks were spent on preliminary research and the remaining weeks were spent on implementing the discussed project. Finally, a report was written and a presentation was given to demonstrate the capabilities of the software system and bring the project to a successful end.

2 Preparation

The project started by conducting two weeks of literature research on the subject and making a project planning for the rest of the project. The results of this research phase are presented in this section.

2.1 Assignment Definition

The assignment is to create a package delivery scheduling system that supports multiple delivery bikes which depart from and return to a central depot. This system should handle several types of requests, several destinations and several (moving) pickup points. The system is further extended by adding time windows, last minute changes and anytime solutions. The bikes that deliver the flowers also have limited capacity, requiring them to return to a depot if they run out. The preliminary research report, which can be found in appendix A, shows that there is a multitude of different ways of both tackling the Selective Pickup and Delivery Problem (SPDP), but also of describing the problem itself. Some papers include Time Dependence (TD) in their problem definition, taking morning congestion into account, for example the paper by Sun et al. [26]. Other problem definitions include several variables to optimise, such as calculating a route with the least amount of distance while also minimising fuel costs. These problems are called Multi Objective (MO) [20]. Other problem characteristics are:

- Partially Dynamic (PD) problems, which allow for the desired waypoints of the route to change while the route is already being driven [28]
- Problems with Movable Pickup Points (MPP) in which the pickup points are dynamic instead of static
- Problems which include Time Windows (TW), which require the vehicle to deliver a package without being too early or late [26]

Combining all of the problem boundaries described above gives us the problem definition: MakeTek wants to be able to prevent congestion (TD), wants to keep customers as happy as possible by delivering flowers on time while also keeping employees happy by reducing the distance they have to cycle (MO). They want to be able to take last minute orders while the delivery process has already started (PD). Vans which serve as pickup points can move position if needed (MPP) and customers are able to specify when they are available at their delivery location (TW).

This results in the **TD-MO-PD-SPDP-MPP-TW** problem: Time Dependent Multi Objective Partially Dynamic Selective Pickup and Delivery Problem with Movable Pickup Points and Time Windows. The problem tackled in this project is interesting because it combines many boundary conditions and features that are not typically found within a single problem. For example, there is a lot of literature about pick-up and delivery problems[1], but they generally do not contain time windows together with dynamic distribution points. Another reason this problem is interesting is because the solution can be used for multiple different purposes. In the case of the system described in this report, the algorithm will be used for flower delivery but it could easily be extended to allow delivery of other goods, such as food or packages.

2.2 Desired Features

The algorithm should be able to calculate routes and schedules for multiple bikes working simultaneously. To make the delivery more convenient for the customer, delivery times should be estimated

[1]A quick search on "pickup delivery problem" on Scopus gives over 1.200 results

and made available to customers. When calculating this, the navigation algorithm should use open source traffic data to take the least congested route to a destination. Deliveries should be distributed over the bikes by clustering on location or time window. Finally, since the bikes have limited capacity the algorithm should make sure the bikes do not arrive at a customer empty-handed by picking up more flowers when necessary.

3 Design Choices

The system consists of two parts: a visualiser and a backend. Both of these parts use different boilerplates based on Node.js and are both written in TypeScript. The visualiser should be run by the customer as a user friendly control system of the backend functionality. The backend should preferably be run on a private external server. The following section gives a more in-depth explanation for these design choices.

3.1 General Design Choices

This section will explain some general design choices made during the research phase. The decision to develop a visualiser as well as a backend is explained first, followed by the decisions concerning the used programming languages and the focus on extensibility. Finally the use of a library for calculating routes is justified.

3.1.1 Visualiser and a Backend

The decision to make two systems, an API (backend) and a visualiser (frontend), was made in the research phase. The client wanted a backend, while the visualiser was not a requirement. However, being able to visualise the solutions a machine learning algorithm returns is very valuable in the development process, especially if those solutions are strings of latitude/longitude coordinates. Building a visualiser which can draw the results on a map allows the team to quickly judge if a solution returned by the machine learning algorithm is 'reasonable' Without such a visualiser this would be either impossible or it would take a long time to rewrite every solution into something a human is able to interpret.

Another advantage of this design choice is that these visual results can be used to demonstrate the functionality and the progress to both the client and the TU coach. The visualiser can also be used to 'sell' the product functionality to potential future clients, who might not have the technical background to judge the backend system in isolation. It also provides the option to build further upon the backend and provide its functionality as a service.

3.1.2 TypeScript

The client requested the program to be written in JavaScript, because the existing software for the flower delivery system was written in JavaScript frameworks as well. The language of choice is Typescript [13], a superset of the JavaScript language that adds type checking and Object oriented programming. Enforcing typing in the code ensures that prevents unintended behaviour. Vanilla JavaScript is not typed, and will run regardless of any 'type' errors. Object oriented programming also allows us to define and extend classes and objects. This keeps the system maintainable and easy to extend.

3.1.3 Extensibility

Another design choice was to keep the system as extensible as possible, so that it remains fairly easy to add or extend functionality in the future. In terms of system design this means that methods and functionality are decoupled where possible and use defined interfaces. Decoupling, as found in the system described in this report, makes sure that every algorithm is built in several 'blocks' which can be swapped. In addition algorithm runners contain an object with all the necessary information to run an algorithm. These algorithm runners can then be run on another computer and report their status back to the main runner. The 'blocks' of the algorithms are defined by what information they receive and what their expected output is. The interface is agnostic to what the block actually does which makes it properly extensible.

3.1.4 Open Source Routing Machine

The system executes a large amount of route calculations. However, because calculating routes is outside of the project's academic scope, an existing implementation for route calculation is used. The Open Source Routing Machine (OSRM) [9] was the best fit. OSRM can be hosted on a company server, and allows us to easily calculate the shortest bike route between several points. In addition, it allows us to calculate a table of durations between all given points. Using OSRM saves a lot of time. In addition, OSRM is likely to work as expected, because the software includes a robust test suite [7] and is widely used.

3.2 Agile Development

For the project the Agile software development method [14] was chosen. This software development method allows for fast changes in the codebase and actively involves the project owner in the design process. This makes sure that you do not spend more than two weeks writing a software system, only to find out that the project owner expected different functionality to be in the end product. It also encourages the developers to deliver a working code base each week, inciting additive development. Following the Agile workflow, sprints of two weeks were planned, including sprint refinement, retrospective and planning meetings with the client, and weekly meetings with the TU coach. For sprint planning the project management tool Phabricator [8] was used.

3.2.1 Sprint Management

Every two weeks a sprint planning and retrospective meeting with the client was held. Normally these meetings are not held with the client, however in the case of this project the client was also the project owner. In these meeting the previous sprint is discussed: "What went well, what can be improved and did all planned features for this sprint get implemented?" Then, together with the client, the initial planning for the next sprint is discussed: "What should we focus on next and which features would the client like to see?"

On top of this a speedboat retrospective session is held. The point of this session is to look back at the previous sprint using the speedboat metaphor. In this metaphor, a distinction between four different categories is made:

- Propellers: what drives the team forward?
- Lifesavers: what can help the team in the future?
- Anchors: what holds the team back?
- Rocks: where can the team crash?

Every team member gets some sticky notes, writes down something about the last sprint and puts it in the right category. For example, if one team member might think that lack of time management is an anchor, he/she writes this down and puts the note under the 'anchors' category.

When every team member has put down his/her ideas the team collectively votes on which ideas are most important to them. Every member gets two votes and the most voted items will get assigned to a person in the team who will then pay extra attention to said item. To continue with the previous example: if the lack of time management item gets the most votes, one person in the team will be responsible to keep an eye on time management by, for example, motivation others when they get distracted.

Halfway through every sprint a refinement meeting is held. In this meeting the team looks back at the first half of the sprint to see how things went and to see if the remaining tickets can be completed on time. in addition, a meeting with the TU coaches is held every week to ensure the academic level of the project. These meetings also allow the team to discuss the implementation of the genetic algorithm and to brainstorm new ideas.

This combination of meetings form an important basis for self-reflection within the team, while also stimulating the team to continuously reflect on itself and to tackle possible problems during the project. This helps in making sure that the team can function optimally, and creates an open environment for raising concerns.

3.2.2 Phabricator

The project management tool Phabricator was used to aid the team with Agile development. Phabricator hosts the git repository where the team pushes code changes and creates tickets with tasks. These tickets are created in the sprint planning phase and give an overview of what needs to be done during this sprint and which functionality is already implemented. Additionally, the Phabricator system allows the linking of these tickets to code commits, and can report test results per commit. (see Figures 2 and 1).

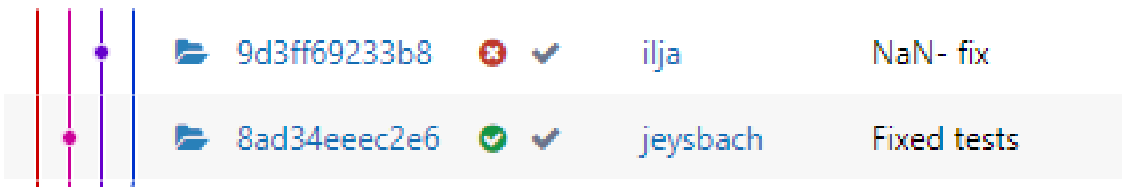
3.3 Quality Assurance

The following section explains the choices that were made to ensure the quality of the developed code is sufficient for use in a professional environment. Firstly, it describes how the code is automatically tested. Secondly, the use of source code analytics is explained. Finally, the implementation of documentation is described.

3.3.1 Continuous Integration and Testing

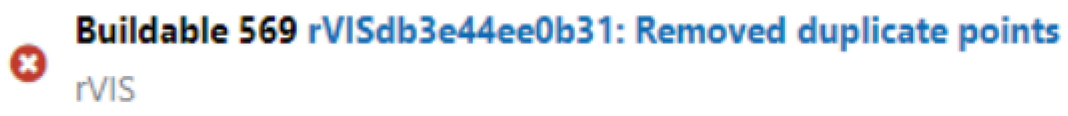
When designing a large system it is important to test the code base and to make sure that already existing functionality does not break when adding new features. This becomes especially important when the system is expected to change drastically, and in the case of this project this is certainly the case. Manually testing the system is possible for a small project, but when the project size increases this generally becomes very time consuming. In such cases test suites which are run every time a code change is pushed to the code base are desired. For both the visualiser and the backend a framework is used which includes Jest [5]. The Jest framework allows us to write test suites as normal TypeScript functions. All test suites can then be run from the command-line with a single command.

Ideally the test suites should be run after every code change. To enforce this a Continuous Integration server was set up, which runs the Jenkins automation server [4]. Jenkins scans for code changes. When a code change is pushed to the server, it pulls the new code, builds it, checks the code using the linter and finally runs all the test suites. If anything goes wrong in this chain, the process exits with an error code and all team members are notified by email with the error log attached (See Figure 3). This ensures that no previous functionality is broken during the design process, with the additional benefit that the tests are run again on a remote server instead of the developer's local machine. This also ensures that the code does not have any dependencies on software not included in the project, i.e. it prevents any 'works on my machine' errors. The build status is also reported back to the Phabricator system, which is then able to show the exact commit that broke the build by marking it with a red `X`. (See Figure 1 and 2)



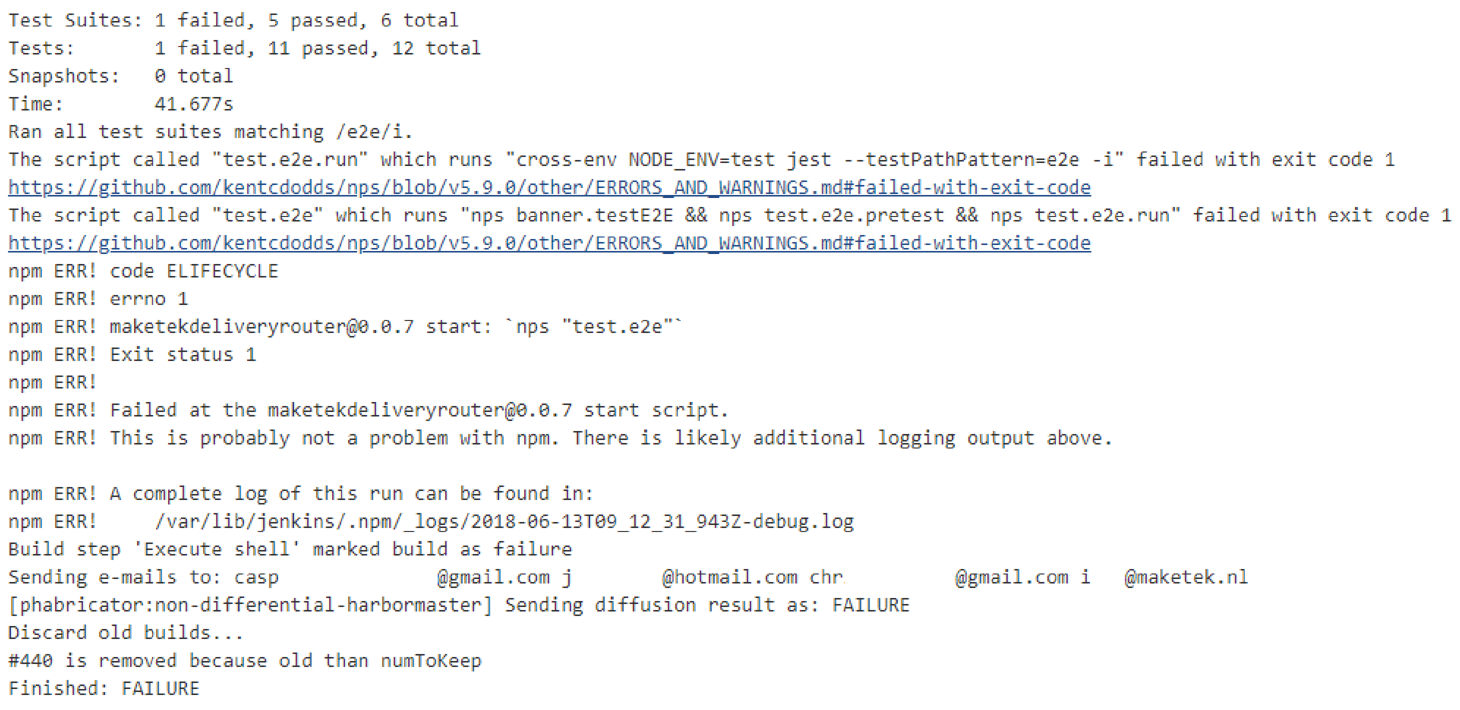
9d3ff69233b8 ✗ ✓ ilja NaN- fix
8ad34eeec2e6 ✓ ✓ jeysbach Fixed tests

Figure 1: Commits for a failing and a passing build



Buildable 569 rVISdb3e44ee0b31: Removed duplicate points
rVIS

Figure 2: Build status in Phabricator



```
Test Suites: 1 failed, 5 passed, 6 total
Tests:       1 failed, 11 passed, 12 total
Snapshots:   0 total
Time:        41.677s
Ran all test suites matching /e2e/i.
The script called "test.e2e.run" which runs "cross-env NODE_ENV=test jest --testPathPattern=e2e -i" failed with exit code 1
https://github.com/kentcdodds/nps/blob/v5.9.0/other/ERRORS_AND_WARNINGS.md#failed-with-exit-code
The script called "test.e2e" which runs "nps banner.testE2E && nps test.e2e.pretest && nps test.e2e.run" failed with exit code 1
https://github.com/kentcdodds/nps/blob/v5.9.0/other/ERRORS_AND_WARNINGS.md#failed-with-exit-code
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! maketekdeliveryrouter@0.0.7 start: `nps "test.e2e"`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the maketekdeliveryrouter@0.0.7 start script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /var/lib/jenkins/.npm/_logs/2018-06-13T09_12_31_943Z-debug.log
Build step 'Execute shell' marked build as failure
Sending e-mails to: casp              @gmail.com j       @hotmail.com chr          @gmail.com i    @maketek.nl
[phabricator:non-differential-harbormaster] Sending diffusion result as: FAILURE
Discard old builds...
#440 is removed because old than numToKeep
Finished: FAILURE
```

Figure 3: Example of a Jenkins log for a failing build

3.3.2 Linting

To keep the code base clean and readable, the decision was made to run a linter on the code base. A linter checks the source code for inconsistent styling and suspicious constructs. The linter used in this project, TSLint [11], can be run from the command line and includes a configuration file which can be adjusted to fit the project. Linting code helps keep the code maintainable, while also ensuring that the programming style is consistent across the entire system to ensure readability of the code.

3.3.3 Documentation

Keeping the code maintainable also means keeping documentation up to date during the development process. The documentation is mostly included in the source code, as all functionality is explained by code comments. The API should also be documented, for which the Swagger library [2] was used. Swagger is a popular API documentation tool, because it can be used to create client and server boilerplate code. The documentation can be accessed via a web page that can be retrieved from the server with an API call.

3.4 Algorithms

The research report concluded that the main algorithm should have a Genetic Algorithm (GA) as its basis. This type of algorithm has been proven to be among the best for this type of problem (see Appendix A: research report), and is relatively easy to modify in order to handle the specific constraints. Before the GA is run, the algorithm should first go through a preprocessing step. This step encodes the real world problem to a format that is usable by the GA. This step can also be used to prepare data by, for example, filtering or clustering. During the project, the decision was made to cluster datasets on their geographical location, with each cluster being handled by one deliverer. By clustering in this way, each instance of the GA computes the route of one deliverer, making the implementation and solutions more intuitive.

In the case of orders changing after the main algorithm is done running, it may not be optimal to run the complete algorithm again. A solution has to be found quickly, thus a greedy algorithm should be implemented. This part of the algorithm should be able to handle added and removed orders. Other problems, like a customer changing time windows or changing the contents of the order, can be modelled as deletions and additions.

Depending on the performance of the first implementation of the GA on the problem, both in the quality of solutions and the convergence rate, other heuristics could be combined with the GA. These heuristics could be implemented in the GA itself, or chained, meaning the output of the GA is used as the input for another algorithm. This can be compared to the way Terada et al. (2006) implemented their GA, which was shown to improve the quality of solutions over a standard implementation of a GA [27]. However, besides the pre- and postprocessing described above, no such heuristics have been implemented due to time constraints and adequate performance of the GA.

3.5 Frameworks

MakeTek proposed two boilerplates to use for the visualiser and the back-end so that the development of desired features could be started as soon as possible. In the following section the boilerplate for the visualiser is described first, afterwards the backend boilerplate is touched upon.

3.5.1 Visualiser

The visualiser boilerplate is based on Electron [3] and React/Redux [25]. Electron is a framework that essentially wraps web pages into an application. This allows programmers to develop their user interface in HTML, CSS and TypeScript, making the development relatively easy. React is a framework that simplifies building user interfaces by ensuring different views are only dependent on the current state. Compartmentalising the application into reusable building blocks in this way makes development and maintenance easier. Redux is a framework that improves on React by introducing actions that can edit the React state and by keeping a global state. Since the state can only be edited by Redux actions, the actions can be seen as a breadcrumb trail of state transitions that makes understanding what is happening inside the application much easier. Another used library is Leaflet [1]. Leaflet is a JavaScript library for interactive maps. The map is used to display routes calculated by the GA, which can serve as a visualisation of the solution.

3.5.2 Backend

The backend boilerplate is based on Node.js.js and TypeScript. Node.js makes it very easy to set up a restful API and TypeScript is an improvement on JavaScript that allows typing and static type checking. Another useful framework included in the boilerplate is TypeORM [12]. TypeORM manages the database interactions.

4 Plan of approach

The plan of approach was formed after research into the requirements. Firstly, together with the client the requirements were put into a MoSCoW model, and used as a foundation for the sprint planning. Secondly, a time plan was constructed to distribute all the requirements over six sprints of two weeks. During the process some requirements proved to be impossible given the timespan and were changed accordingly. This is highlighted in the final adjustments section.

4.1 MoSCoW

MoSCoW is a technique used to prioritise requirements in a software engineering process. It is an acronym of the phrases: ”Must have”, ”Should have”, ”Could have” and ”Won't have” [15].
This section contains the prioritisation of the product's features that was made during the beginning of this project. It does not contain a *Won't-haves* section, due to all encountered applicable features initially being considered by the team. This is a result of the relatively short planning phase in which the difficulty of features could not be fully estimated.

4.1.1 Must-haves

The final product must have at least one algorithm that converges to a (local) optimum. Without this requirement the algorithm would be unusable in practice. It must also be able to dynamically adapt to changes like deliveries being added. It must also be able to dynamically adapt to changes such as shifting time windows and last-minute additions. Another must have is that the algorithm should be able to provide a solution at any time. This ensures that the start of deliveries is not bounded by the runtime of the algorithm. To make sure a multitude of edge cases are tested, datasets should be a random subset of realistic deliveries. There must also be a possibility to link the real datasets of the customer with the final product.

4.1.2 Should-haves

A visualiser should be implemented to be able to showcase solutions to the client. The visualiser also helps with debugging, since it is easier to spot unexpected behaviour when the routes and schedules are drawn on a map. The algorithm should also be able to support live arrival times. Finally, the bikes have limited capacity, which should also be represented in the algorithm.

4.1.3 Could-haves

The first could-have is using statistics from traffic data to determine the routes. This is a could-have because writing a routing algorithm is not within the scope of this project. Another could-have is that the positions of the depots where bikes can pick up flowers is dynamically determined.

4.2 Time Plan

The project was divided into 5 sprints of two weeks. Every week consists of several tickets which are divided between the team members to create an equal division of labour.

- **Sprint 1 (Research Report)** In the first sprint research into existing solutions is done and an initial algorithm is designed. The initial set of requirements (MoSCoW) and a feasibility analysis are also made in collaboration with the client during this sprint. A plan of action is made from the requirements, and finally the basic project structure is set up. The conclusions of the research are collected in the research report (see appendix A)

- **Sprint 2 (Prototypes)** The basic data structures will be designed and implemented in this sprint. This includes the data provided by MakeTek. A dummy dataset is provided that is representative of real life data. The deliverable is a basic web application for display and demo purposes. It should be able to retrieve data from the server and be able to show an initial data model on the map.
- **Sprint 3 (Algorithm)** In sprint three a first version of the scheduling algorithm is implemented. It should work with at least one bike. Secondly, a Routing provider is installed to provide distance data between points. The deliverable is a first demonstration of the results of the algorithm.
- **Sprint 4 (Refinement)** After receiving feedback a new sprint is started to improve the first version of the algorithm with new features. The algorithm pipeline is also extended with different algorithms and parameter options, and different algorithms and heuristics are combined. The routing is also improved to include delivery schedules based on maximum capacity. The deliverable is a second iteration of the scheduling algorithm.
- **Sprint 5 (Report)** In this sprint the final adjustments are made to the algorithm including delivery delay and optimisation using advanced heuristics. Preparations are made for the production release of the system. The main deliverable this sprint is the final report.
- **Sprint 6 (Production)** In the final sprint the team presents their findings and their product by means of a presentation and a live-demo. In addition, the code is finalised for production.

4.3 Adjustments

During the project some features were scrapped from the final design. These scrapped features, together with the justification to scrap them, are described in the following sections.

4.3.1 No Congestion Control

An external program was needed to calculate the routes between points. OSRM was a free and relatively easy to integrate option. However, this program does not automatically incorporate live traffic data in the route calculation. It is possible to adjust it to do so, however it has been deemed to not be worth the time investment. On top of that, bikes suffer less from congestion compared to cars.

4.3.2 No Hyper Heuristics

Initially the idea was to add heuristics on top of the implemented algorithm (hyper heuristics) to determine the best input variables. This would result in an adequate set of variables and therefore a good set of results. The decision was made not to do this because of time constraints. Properly constructing hyper heuristics is a small project of its own and is outside the scope of this project.

4.3.3 No Dynamic Pickup Points

Dynamic pickup points were dropped, because the overhead of checking the effectiveness of a certain position would be too computationally heavy. On top of that the fitness of the routes the deliverers take is dependent on the location of the depot. This means dynamic depot locations either need to be integrated into the determining of the routes, or the algorithm has to be run multiple times. The alternative was to cluster the order locations and then use the centroid of each cluster as the pickup point. While this does not result in the same quality of results as the dynamic pickup points, it saves computing time and complexity, as well as giving us enough time to implement the features that were prioritised by the client.

5 Final Design

The following section contains details about the final design of our product. Firstly, the visualiser is described using a Visual representation of the system architecture (see Figure 4). Secondly, the same is done for the backend (see Figure 5). In the next section details about the implementation of the algorithm in the final system are described. Then, an overview of the API endpoints is given. After this, the implemented datamodel is discussed. Finally, the implemented tests and their results are described.

5.1 Visualiser

The visualiser starts with the home page. From this page the other pages can be reached. There are three other pages: the MapComp, the StateManager and the GAController:

- The MapComp page contains a map component that is constructed by leaflet. On this map the solutions from the backend are drawn. The deliveries are marked with a colour depending on whether they are on time.
- The StateManager page allows management of displayed routes. It is possible to remove routes and their points and it is possible to run a route query on OSRM directly. This page also displays some more information about the displayed routes.
- The GAcontroller page contains a form that allows the user to select from all the configurations the backend provides. The form is generated dynamically according to a configuration file that is retrieved from the backend. The GAController also allows downloading started tasks and their results from the backend.

Every page has its own state, containing the variables that are relevant for that page, and actions that manipulate its state. However some pages can influence states of other pages as well. The pages use calls through the functionality classes to the HttpRequester, which is used to handle all requests to the back-end.

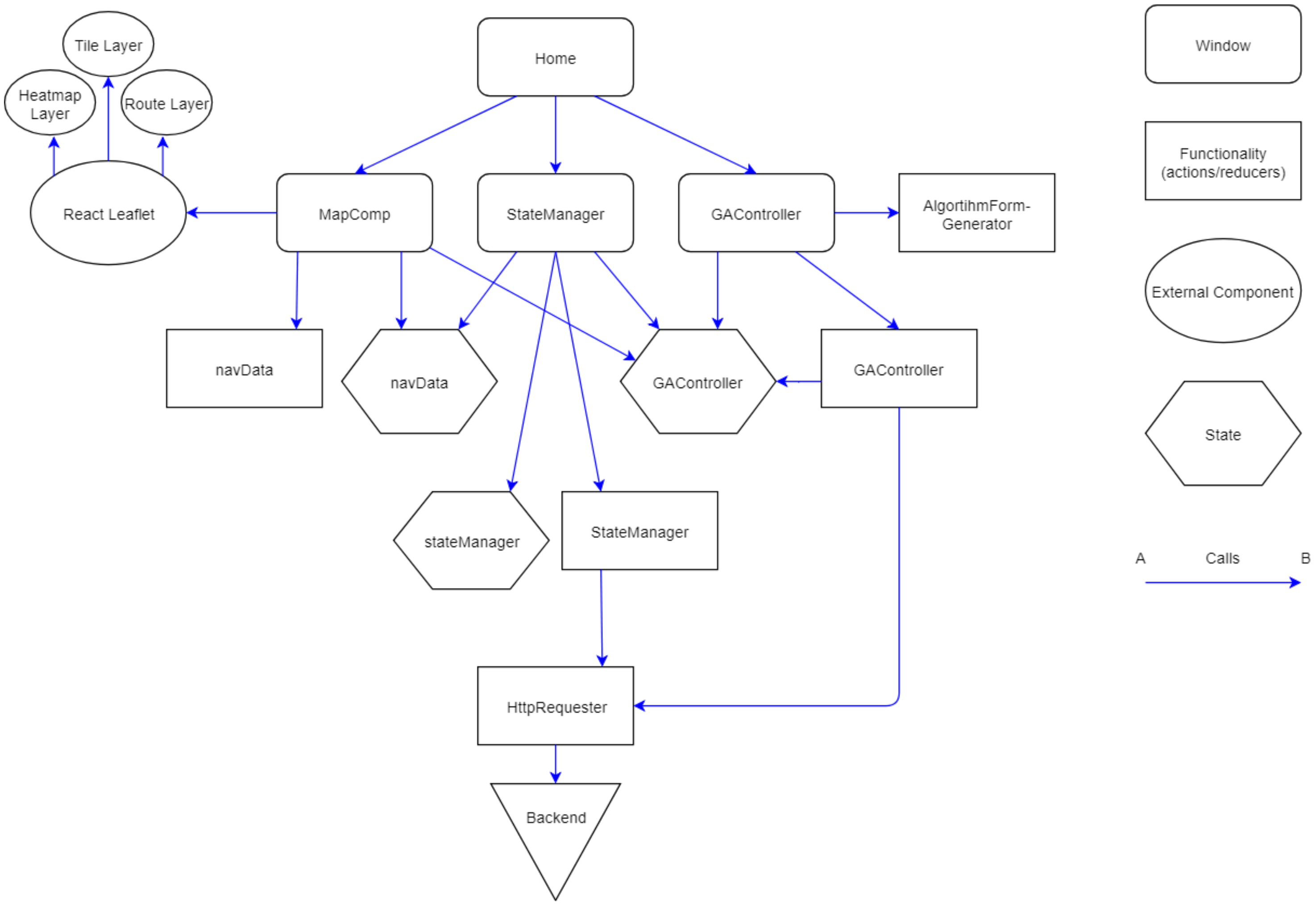


Figure 4: Visualiser architecture diagram

5.2 Backend

The backend is accessible by the visualiser through the API middleware. API controller functions are called through CRUD (Create Read Update Delete) HTTP requests. These functions call backend services, which are handlers for executing server functionality. Within these functions interaction with the database might be required, the service functions do this by calling repository functions that are responsible for data storage and retrieval. Because the controllers represent the functionality of the server best, the most important controllers are highlighted:

- The DatasetController and TaskController handle the retrieving of datasets and tasks respectively. A task represents a routing problem, and datasets contain all deliveries that need to be carried out in a Task.
- The AlgorithmController can be used to manage run configurations of available algorithms. This was added because of the focus on extensibility in the system, however it is currently unused in favour of a JSON file that contains the algorithms with all their required variables.
- The SolverController starts, stops and returns the status of instances of the algorithm.

The algorithm part of the system consists of three pieces: **preprocessing**, **main algorithm** and **postprocessing**. The **preprocessing** part is used for clustering within the final system. These clusters are made through the SolverService which divides the dataset into the right amount of clusters and launches the genetic algorithm on each of the clusters separately.

The **main algorithm** part runs the genetic algorithm. Its parts are all interfaces with multiple implementations, allowing the user to choose which implementation to use. When the main algorithm returns its solution to the SolverService, a postprocessing step can be started.

The **postprocessing** contains a last-minute insert option, which uses a greedy algorithm to handle real-time additions. Postprocessing also decides when the deliverer has enough time to take a break.

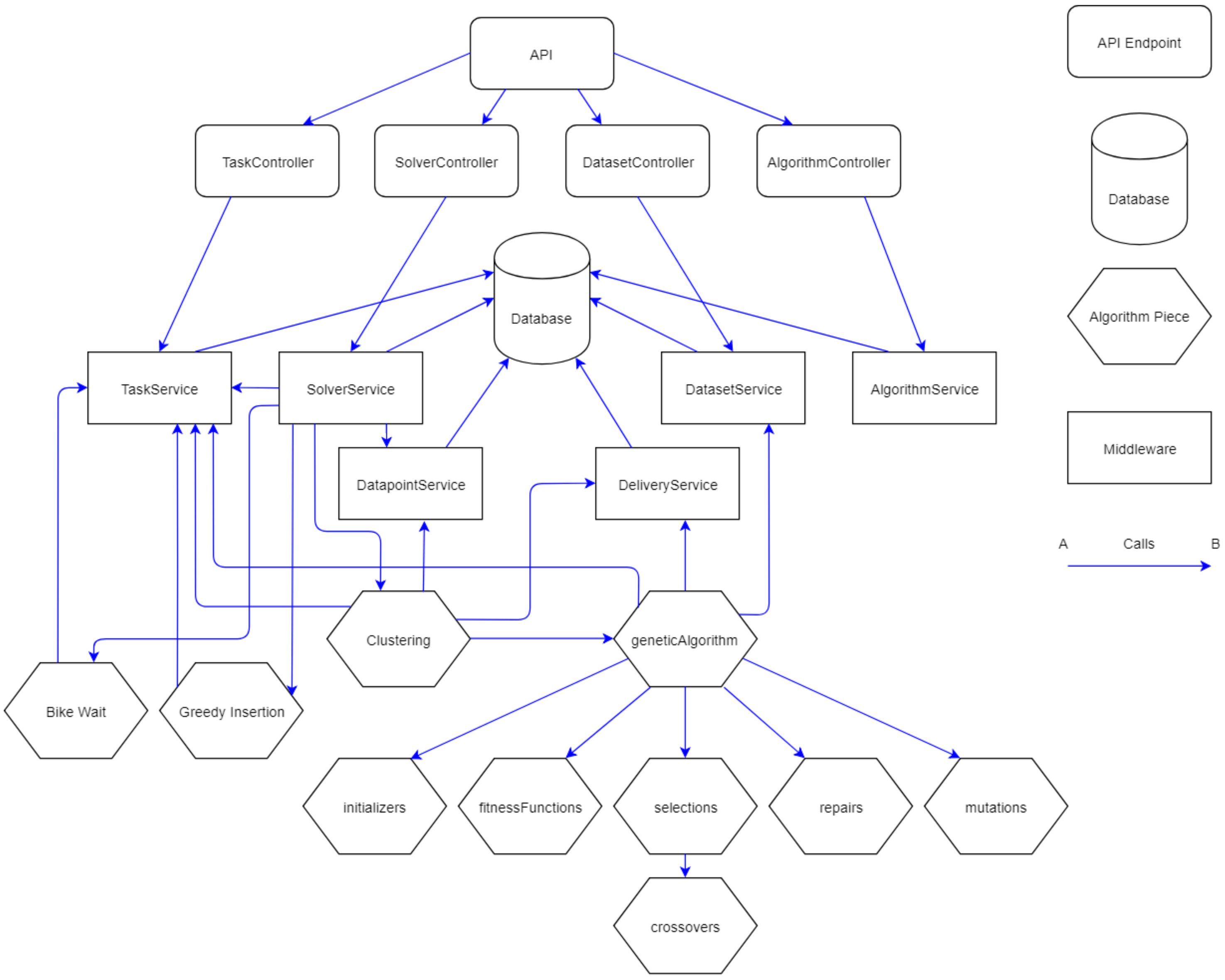


Figure 5: Backend architecture diagram

5.3 Algorithms

This section contains details about the implementation of the various algorithms used in the back-end. The clustering is explained first, followed by an in-depth look at the body of the genetic algorithm. Finally the postprocessing is discussed.

5.3.1 Clustering

In order to create a usable set to run the genetic algorithm on, the datasets are clustered and used as separate instances for the GA to solve. The final implementation contains k-means [22] and fuzzy c-means clustering [16]. The difference between these two clustering algorithms is that c-means clustering allows points to belong to several clusters, while in k-means clustering every point can only belong to one cluster. C-means clustering was implemented to prevent possible sub-optimal solutions when points lie on the edges of a cluster. In such cases it might actually be better to assign a point to the neighbouring cluster if this improves the general quality of the solution instead of sticking to the hard limits provided by k-means clustering.

The clustering can be used on: geographical location, travel distance or time-windows. In the implementation discussed in the report deliveries are clustered on geographical location. Clustering was implemented in this way so that every cluster of deliveries belongs to exactly one deliverer. It also prevents other problems that would occur with the routes if the datasets if the datasets are clustered per time window. Namely, if a deliverer has to work through multiple clusters, two new problems arrive: the first is the correct assignment of the succeeding clusters to the deliverers, and the second is the alignment of the last delivery in the preceding cluster to the first delivery in the succeeding one. A potential improvement would be to take time windows into account in the clustering with a higher level algorithm control to prevent said problems.

Another approach is to encode the clustering in the solution, and use the GA to find the clusters during run-time. However, the choice was made to cluster beforehand by preprocessing, because the other option would most likely decrease the convergence rate and increase the complexity of the GA. Another advantage of the preprocessor approach is that it can be further optimised for different scenarios.

5.3.2 Genetic Algorithm

The main body of the algorithm is a GA with interchangeable parts, meaning every step of the algorithm can be replaced by a different implementation. The GA starts by initialising the first generation of solutions, with the size of the generation defined by the user through the API call. After initialisation, the fitness of the solutions in this generation is determined by checking how good a solution is based on a fitness function. Then, for the desired amount of generations, which is also set as a parameter in the API call, the algorithm does the following:

1. Selection
   The aim of selection is to determine which solutions in a generation are responsible for the solutions in the next generation. To accomplish this the algorithm selects two solutions (parents) from the current generation based on their fitness. Selection is what allows a GA to converge to an optimal solution, because solutions with a high fitness have a higher chance of producing offspring.

2. Crossover
   Crossover is a form of propagation to create the solutions for the next generation (children) by combining the genomes of the chosen parents.

3. Mutation
   To prevent getting stuck in a local minimum, mutation is performed. This randomly changes parts of the solutions with a given chance to explore the solution space.

4. Repair
   In the case of delivery problems, the combination of crossover and mutation almost always leads to invalid solutions. To make sure solutions stay valid, and to be able to produce a result at any time, the solutions should be repaired.

Time windows are seen as a part of the problem which should be optimised, because not arriving on time does not invalidate a solution. Being on time can also have a lower priority than the duration of a deliverer's route. Hence, the deviation from the time windows is encoded into the fitness function.

All the implementations of the parts described above and their parameters are described in the following subchapters. Most of the parameters that should be provided to the GA are choices between different implementations of the above mentioned parts. The other parameters are mentioned in their respective sections.

5.3.2.1 Encoding

To run a GA on the problem, a solution encoding first has to be determined. All the steps of the GA should be able to work with this encoding. The implementation discussed in this report uses the order of deliveries as a solution. For example: `[4,2,1,3]` means visit customer 4, 2, 1 and 3 in that order. This is called **permutation encoding** [17].

The problem described in this paper has two more constraints that have to be dealt with, namely the delivery bike capacity and the time windows of the deliveries. Capacity is handled in the encoding by adding a `-1` to the sequence to represent a trip to a depot.

5.3.2.2 Initialisers

There is only one type of implemented initializer. It creates the desired amount of solutions and then applies Fisher-Yates shuffle [19] to randomize the order of each solution. This results in a completely random initial generation. This choice was made to make sure the algorithm explores the solution space properly. The currently implemented initialiser takes the desired generation size as a parameter.

5.3.2.3 Fitness Functions

There are two implemented fitness functions. Both use the following formula to decide the fitness:

$$fitness = a \times \frac{1}{dur} - b \times tw \tag{1}$$

with a being the duration weight, dur being the route duration in seconds, b the time window weight, and tw being the total breach of time windows, i.e. the total amount of seconds the deliverer is too late or early at his/her deliveries. The reason equation 1 is divided by the duration is that low duration means a better solution. This means the ratio between the duration weight and time window weight is not 1:1, and the duration weight should be higher to properly balance the weights. The duration weight and time window weight are parameters that have to be supplied in the API call.

The difference between the two implementations is that one also uses the validity of a solution (whether it contains duplicate deliveries / is missing deliveries) as a value in the calculation as follows:

$$fitness = f + c \times (100 - \frac{100}{n} \times x) \tag{2}$$

with f being the fitness described in equation 1, c being the validity weight, n being the solution size and x being the amount of duplicates in the solution. Equation 1 should only be used in combination with a repair function, otherwise the optimal solution will not be a valid one. Equation 2 is intended to be used without a repair function.

5.3.2.4 Selections

Initially, roulette wheel selection was implemented, because it is relatively easy to implement. In this type of selection, the chance a solution is chosen as a parent is: $\frac{fitness}{totalfitness}$.

During the project, the choice was made to implement tournament selection, due to it having better solution quality with lower runtime than roulette wheel selection in similar problems [24]. Tournament selection takes k solutions from a generation as a tournament. The solution with the highest fitness in each tournament is selected as a parent. This is repeated until enough parents are chosen. The parameter k is provided through the API call.

5.3.2.5 Crossovers

Three types of crossover have been implemented. All functions aim to create offspring based on the genomes of two parent solutions.

- Single point crossover takes a random index x between 0 and the length of the shortest parent. Child 1 gets parent 1's genes from index 0 to x, and genes from index x to the end of the genome of parent 2. Child 2 gets genes 0 to x from parent 2, and the genes from index x to the end of the genome of parent 1.
- Two point crossover takes two random points between 0 and the length of the shortest parent. Child 1 gets parent 1s genes up until the first point, parent 2s genes up until the second point and parent 1s genes after. Child 2 gets the other pairs of genes, as is seen in figure 6.
- In Uniform crossover, child 1 gets each gene from either parent with a 50% chance and child 2 gets the gene from the other parent.

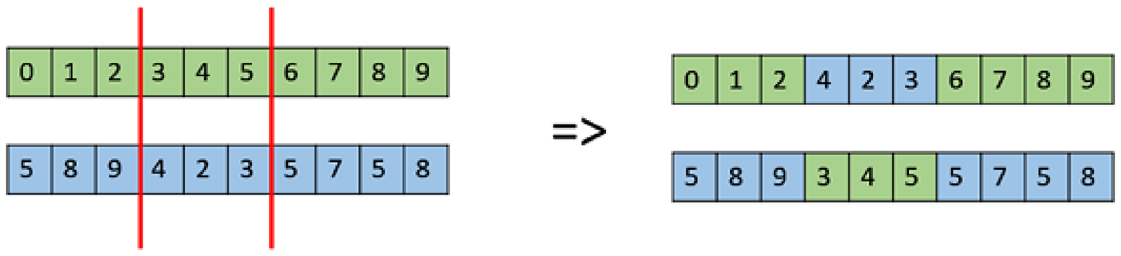


Figure 6: Example of two-point crossover [2]

[2]Retrieved from: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

5.3.2.6 Mutations

Two types of mutation are implemented. Each solution has a defined chance to mutate. Two-opt Mutation swaps a random delivery in a solution with another random delivery in the same solution. N Two-opt Mutation does the same as two-opt mutation, but performs these swaps n times, with n being a random number between 0 and the solution size. However, to ensure the GA finds an optimal solution with regard to depot visits, it should also explore the optimal position for these. Therefore, mutation also has a chance to add or remove a depot from a solution. Mutation requires a single parameter: the mutation chance. It is the chance that the mutation is executed on a solution.

5.3.2.7 Repair

One type of repair function is implemented. It removes duplicates and replaces them with random unused deliveries so that the solution is valid again. It adds the rest of the unused deliveries in random spots. The reason they are added randomly is to keep the runtime to a minimum, prevent the forming of biases and to further explore the solution space. Repair also removes consecutive visits to the depot, i.e. `[1,2,-1,-1,-1,3]` is repaired to `[1,2-1,3]`. The repair function also adds a visit to a depot if the bike does not have enough flowers for the next delivery.

5.3.3 Postprocessing

After the GA is done running, postprocessing can be applied to adjust the solution to match a certain preference. This section describes two types of post processing: addition of deliveries and the possibility for the deliverer to take breaks.

5.3.3.1 Addition of deliveries

One of the requirements of the system is that it should allow the dynamic addition of a delivery into an existing solution. For this two greedy algorithms were implemented that focus on different parts of the constraints:

- The first type of greedy algorithm takes into account the difference in duration when inserting the new delivery. It loops over all possible positions and calculates the difference in duration of the solution when placed there. It then inserts the delivery in the position where the difference in duration is minimal.
- The second type of greedy algorithm is based on the fitness. It checks all available positions for the new delivery and determines which one gives the best fitness using the same fitness settings as the original task.

5.3.3.2 Breaks in the Deliverer's Route

The main genetic algorithm only returns an ordering of deliveries and the durations of traveling between these deliveries. This means that it still has to be determined when exactly the bike will start delivering and when it has to wait because it would be too early for the next delivery. While the deliverer is waiting, he/she could sell the excess flowers in his/her bike, or take an extra break. However these breaks are not mandatory. The best way to determine when to take these breaks is in a greedy way in post processing, as calculating this as an integral part of the GA would be too computationally expensive.

5.4 API

A stateless Restful API forms the base of the interaction with the system. Because the algorithms take a long time to complete for higher amount of generations, a stateless service provides more consistency than working with sessions. The API interactions are based on:

- Algorithms: Lists a set of algorithms and their expect parameters. This is used to create a model based UI on the visualiser, where the interface is defined by the model of the algorithm.
- Datasets: The datasets endpoint gives the options to upload new datasets or view existing, the back-end of the clients system can upload the datasets in advance before starting the algorithm, giving the routing service time to create a full distance table between all points.
- Tasks: The backend can track the status of all running tasks by polling them in the database, and can start/stop tasks manually.
- Route: The clients delivery tables can approach this endpoint to get their delivery nodes and their current best route. This way they only receive the data that is useful for them.

5.5 Datamodel

The implementation of the first version of the system focused on practical implementation choices. The system now stores the data model in a MySQL database, which offers a good balance in speed versus storage. Further versions can easily implement different database structures, such as a NoSQL database [18] for the large objects of taskresult, and a key-value storage system such as Redis [10] to store distance tables in memory for quick access. Swapping to different databases is easy because the system uses a database agnostic ORM, meaning it works independent of what kind of database is used.

As shown in Figure 7 the current datamodel's focus is on extensibility, with a Dataset that is easily adaptable by the user, the option to store relevant fitness parameters in the task, and the low storage identifiers of the data points for the GA.

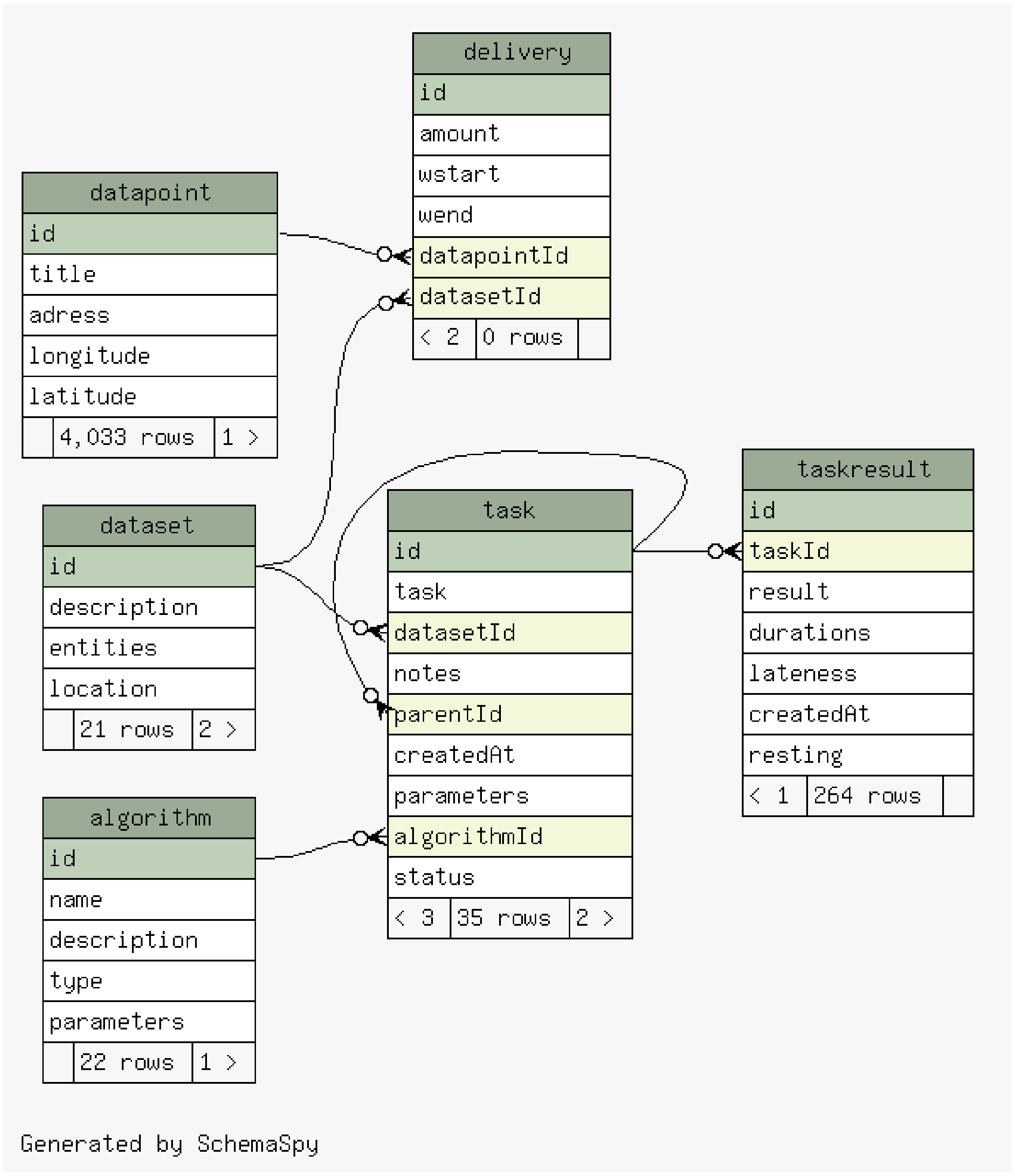


Figure 7: Database schema as used in the final implementation of the system

5.6 Test suites

The final code deliverable contains several test suites which test the functionality of the code. The visualiser includes 9 unit test suites which add up to 60 unit tests in total. The coverage report of these unit test suites, generated by Jest, can be found in Figure 23 in Appendix C. The visualiser also includes a test suite with 21 end-to-end test cases which verify the behaviour of the user interface elements.

The backend contains 18 unit test suites with 70 tests in total. The code coverage of these unit tests can be found in Figure 24 in Appendix D. Additionally, the backend contains 4 test suites with 9 larger tests in total to check database interactions. The coverage of the integration tests can be found in Figure 25 of Appendix D. Finally, the backend includes 6 test suites with 13 end-to-end tests which verify the behaviour of the system by making an API call, waiting, and then checking if the returned results are valid when requesting them via another API call. The coverage of the end-to-end tests can be found in Figure 26 in Appendix D.

The overall statement coverage of both the backend and the visualiser is above 80%. Code is unit tested where possible, but in some cases, like database interactions, this functionality is tested in an integration test instead. Finally, end-to-end tests ensure that the system behaves as intended.

6 Results

In this section the results of running the GA on different datasets is described to demonstrate its performance. The GA implementation allows for several parameters to be set, and thus the goal of this section is to check how these parameters influence both the quality of the solutions and the rate of convergence.

6.1 Approach

The collection of data used in this section was done by running the GA five times using fixed datasets. Fixed datasets were chosen because content of the datasets can have a significant impact in the final quality of the solution. This enables studying the effect of parameters in isolation and seeing how they affect the solution. The datasets used for these results are shown in figure 8, 9 and 10 and are described below:

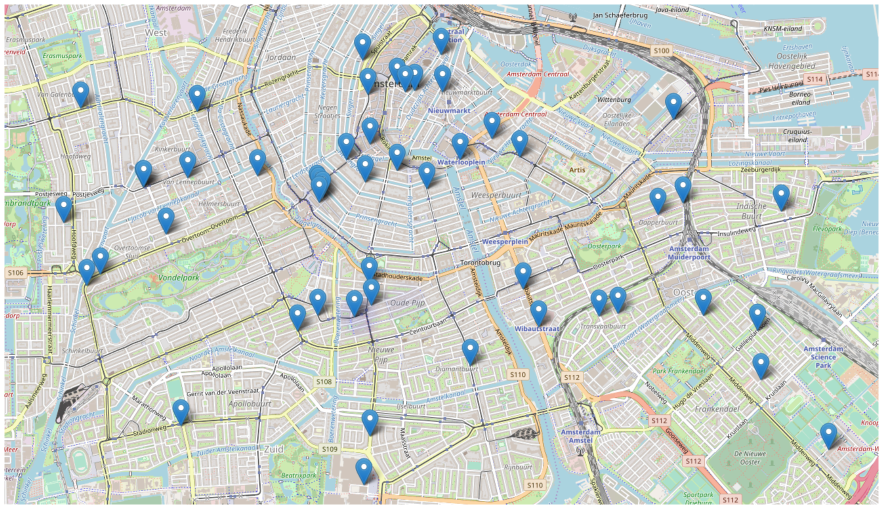


Figure 8: Dataset with 50 points

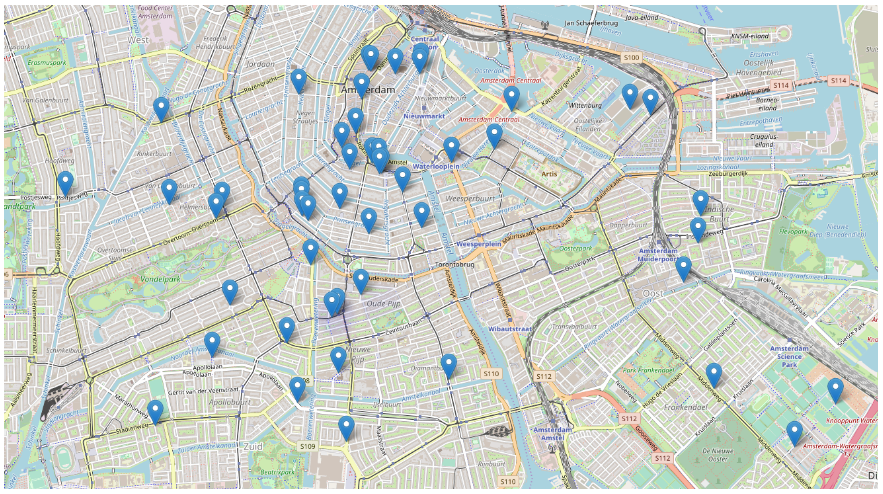


Figure 9: Dataset with 50 points that tend to be in close pairs

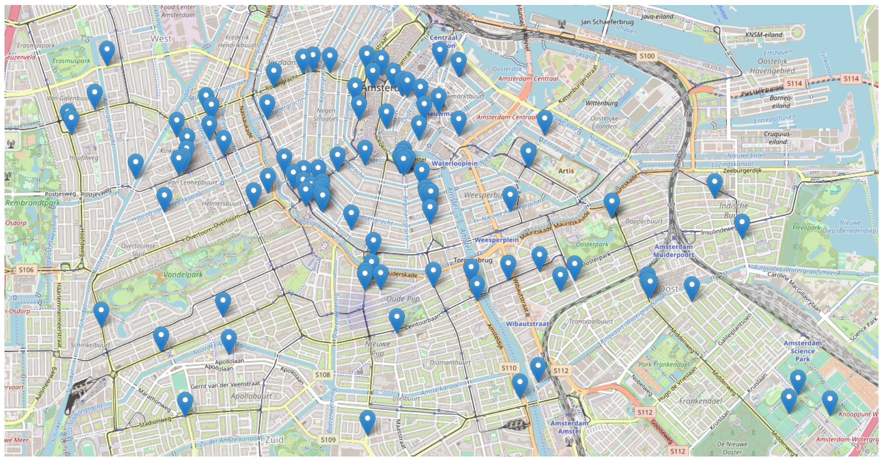


Figure 10: Dataset with 100 points

- Dataset A (`p0close50noTW`, see Figure 8) includes 50 points uniformly distributed over the centre of Amsterdam, and has time windows which last from the beginning of the day to the end of the day.
- Dataset B (`p2close50noTW`, see Figure 9) includes 50 points distributed in tuples (two points close together) over the centre of Amsterdam, and has time windows which last from the beginning of the day to the end of the day.
- Dataset C (`p0close100noTW`, see Figure 10) includes 100 points distributed uniformly over the centre of Amsterdam with time windows lasting the entire day, and is mainly used for measuring clustering performance.
- Dataset D (`p0close50TW`, see Figure 8) includes 50 points distributed uniformly over the centre of Amsterdam, with a random distribution of hour long time windows, used for measuring the effect of weights on the solution.

6.2 Effect of Parameters

In this section the effect of the parameters on the final fitness and the rate of convergence is shown. For all results it holds that the amount of generations was set to 5000, the generation size was set to 100, the selection parameter was set to tournament, the tournament population size was set to 5, the mutation chance was set to 0.3, the mutation type was set to NTwoOptMutate and the type of crossover was set to TwoPointCross, unless specified otherwise.

6.2.1 Mutation Parameter

The effect of the mutation parameter is tested on both dataset A and B. Time windows are chosen as large as possible to study the effect of mutation independent of other possible influences on the fitness.

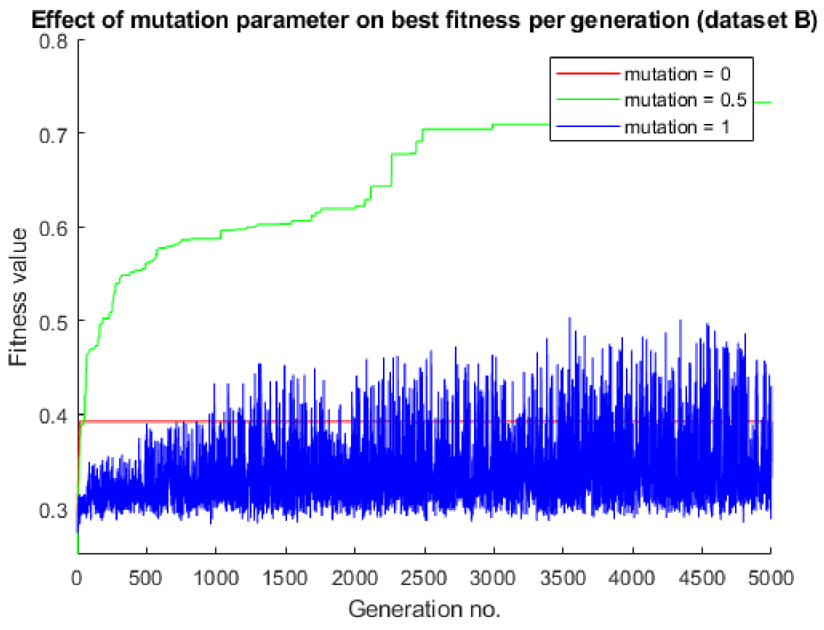


Figure 11

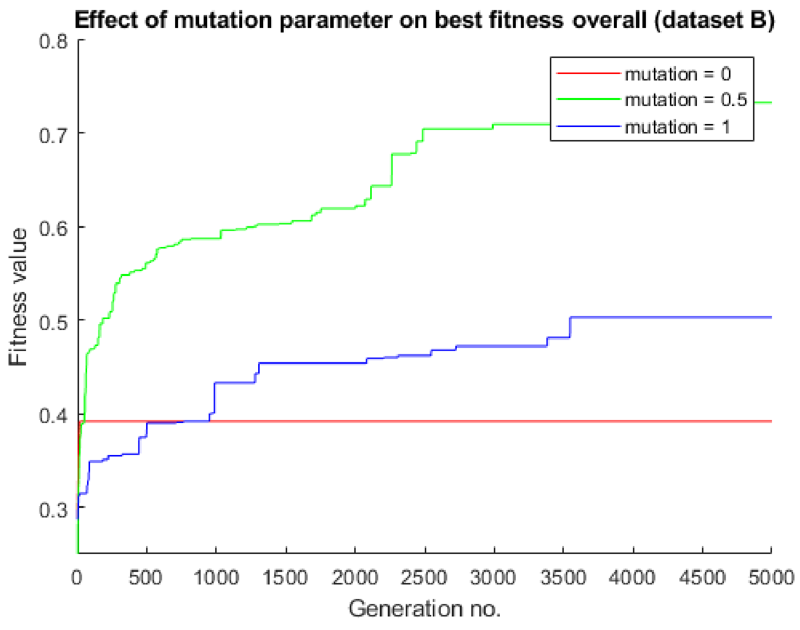


Figure 12

The results for running the GA on dataset B show that setting mutation chance to 1 gives fitness values which appear close to random. (Figure 11) Sometimes by chance it hits a better fitness value, which is then stored (Figure 12). Because the best fitness is stored, the solution converges in 'jumps' when by chance the GA has discovered a better solution. The results of setting mutation chance to 0, however, shows that the GA very quickly finds a local optimum and gets stuck there, unable to explore the solution space. Setting the mutation chance somewhere in between these two extreme cases (in this case to 0.5) allows the GA to explore the solution space, while also steadily converging.

The influence of the mutation chance on dataset A was very similar, with a mutation chance of 1 being marginally better than a mutation chance of 0, and a mutation chance of 0.5 performing significantly better. Setting the rate somewhere in between (0.3 - 0.7) allows the algorithm to converge on both datasets while also properly exploring other optima.

6.2.2 Selection Parameter

The effect of the selection parameter is tested on both dataset A and B. Time windows are again chosen as large as possible to study the effect of selection independent of other possible influences on the fitness.

Tournament selection appears to perform better than Roulette selection for the dataset A and B, at least when running 5000 generations. Figure 13 shows that the best fitness per generation for roulette selection is more volatile than the fitness per generation for tournament selection, however in figure 14 it is shown that both selections converge towards better solutions, although roulette does so at a slower pace than tournament selection.

Running the same tests on dataset B showed equal results, however both solutions have higher fitness values due to deliveries being closer to each other in this dataset. When run on dataset B roulette selection converged slower than tournament selection, and roulette showed the same volatility.

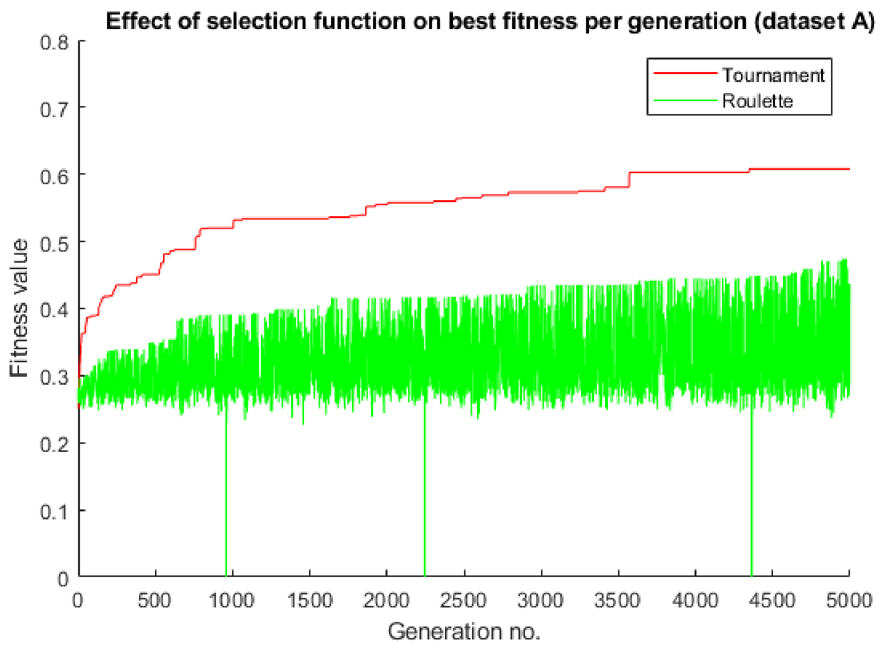


Figure 13

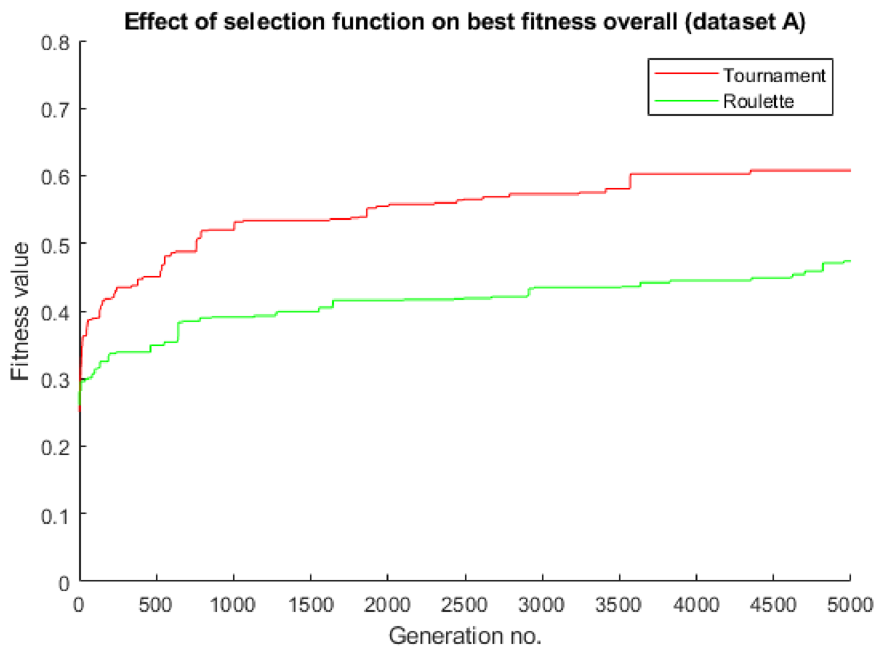


Figure 14

6.2.3 Crossover Parameter

The effect of the crossover parameter is tested on both dataset A and B. Time windows are again chosen as large as possible to study the effect of crossover independent of other possible influences on the fitness.

The difference between the crossover functions for the datasets appears to be minimal. In some runs one of the crossover functions appears to perform better than the others, while in other runs this crossover function performs the worst of the three. This was the case across both dataset A and B. An average graph for the differences can be seen in figure 15.

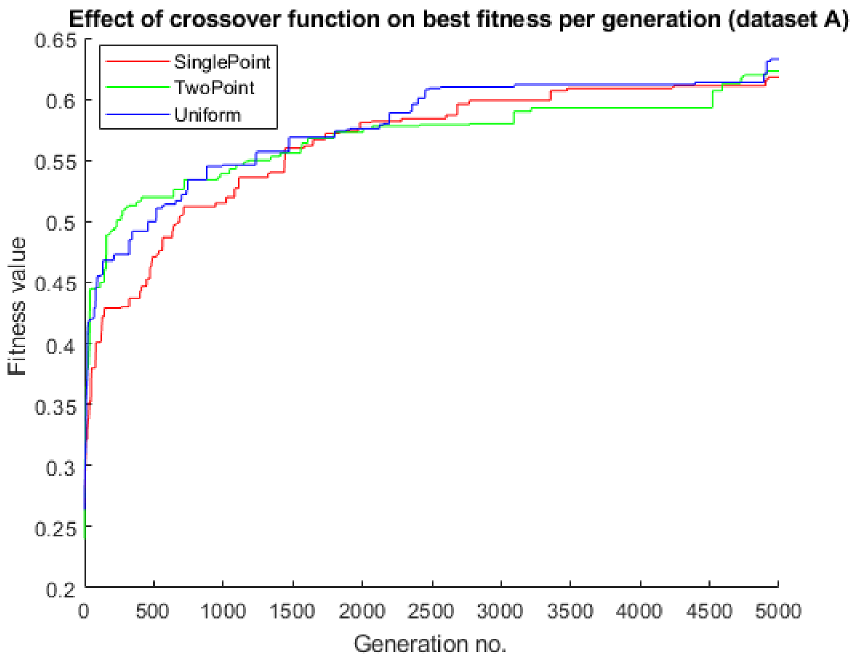


Figure 15

6.3 Effect Of Clustering

The performance of the clustering algorithm is tested on dataset C. 100 deliveries are divided over four different deliverers using the K-means clustering algorithm as described in section 5.3.1 with k set to 4. The resulting clustering and route calculation is shown in Figure 16.

The results show that the cluster edges are hard edges, with some points lying on the boundary of two different routes. In such cases it might actually be preferable to have such a point belong to the neighbouring cluster if this improves the final route. For this, instead of running the K-means clustering algorithm, the Fuzzy C-means clustering algorithm can be run, as described in section 5.3.1.

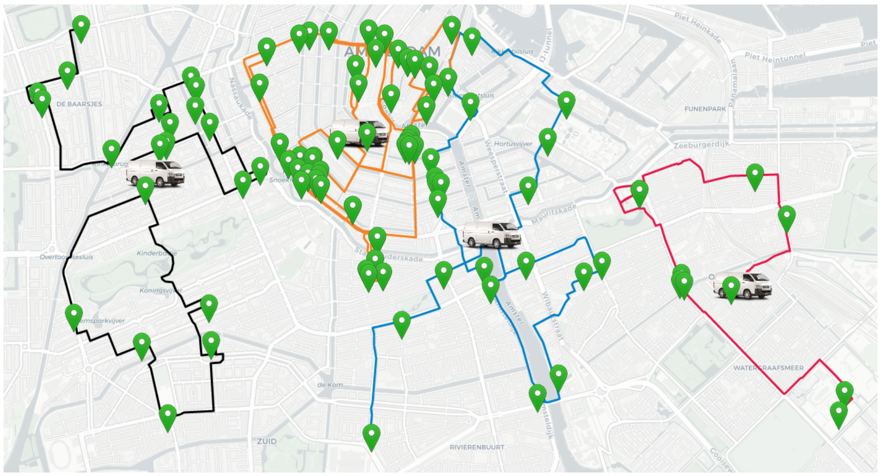


Figure 16: A solution given by the GA using K-means for k=4

6.4 Effect Of Capacity

The effect of capacity was tested on dataset A. When running the GA with unlimited bike capacity, the route in Figure 17 is the result. However, when setting the maximum bike capacity to 10 and letting one deliverer do 50 deliveries, it is clear that the route becomes longer because of refill trips to the depot. In Figure 18 such a route is shown.

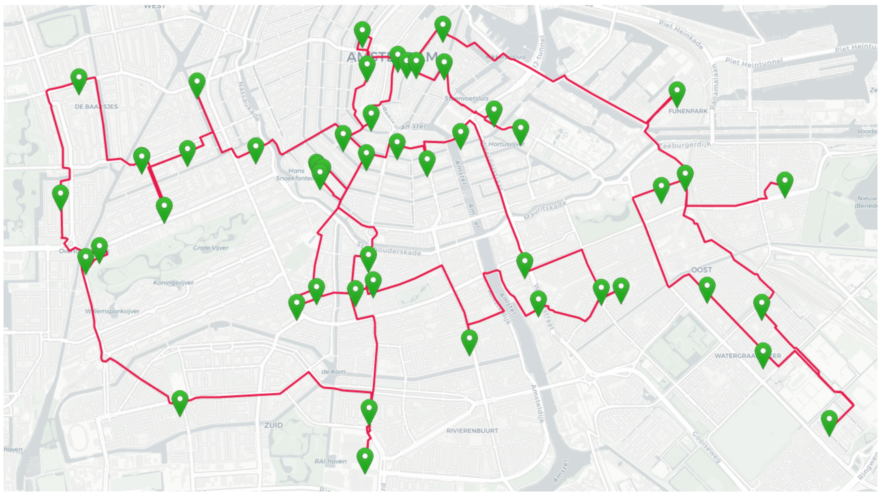


Figure 17: 5000 generations of the GA using unlimited capacity

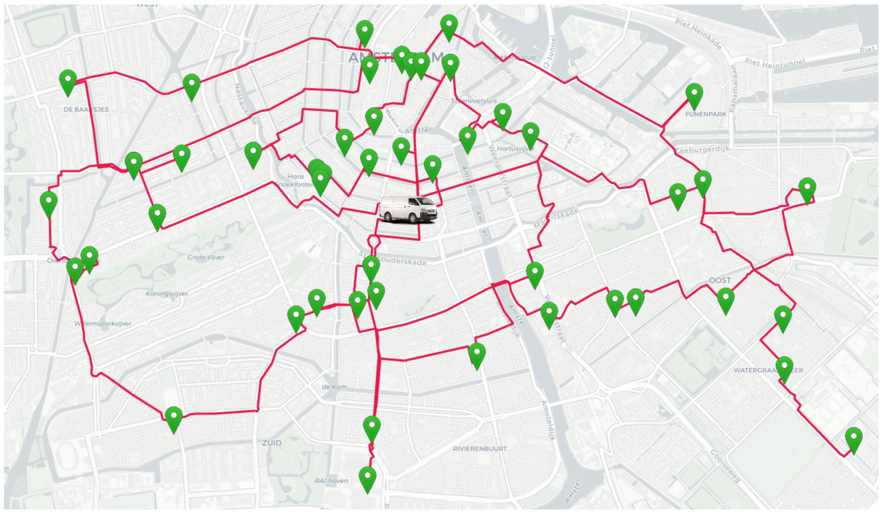


Figure 18: 5000 generations of the GA using a capacity of 10

6.5 Effect Of Weights

The effect of weights was tested on dataset D. The effect of the weights as described in section 5.3.2.3 on a run of the GA is easily visible. In Figure 19 the weight for the time windows was set to 0, while the weight for the duration was set to 100. As expected, the route is minimised on the duration and thus a lot of deliveries are either too early or too late, but the route is relatively short. Figure 20 shows a run with time window weight set to 100, while the duration weight is set to 0. It is clear that in the second case a lot more deliveries are on time, however the route is noticeably longer. The balance between these two weights depends on the preferences of the user, the nature of the dataset and the amount of deliveries per bike.

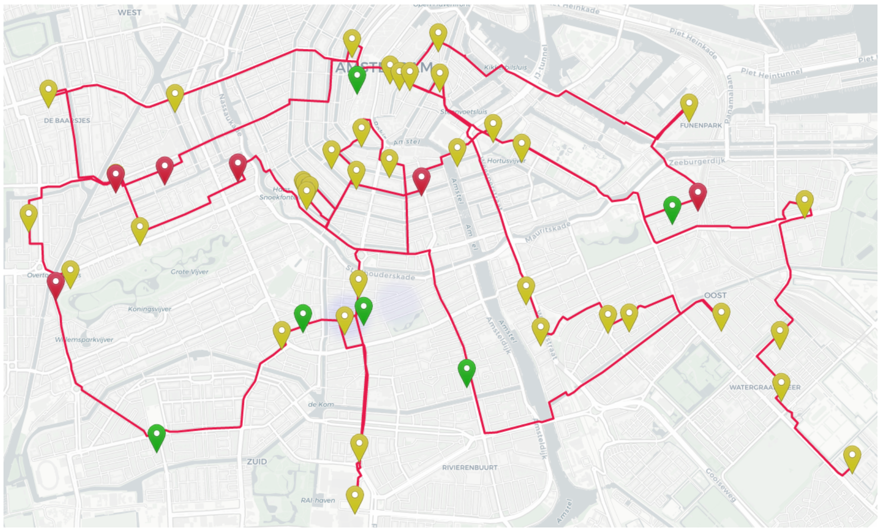


Figure 19: 5000 generations of the GA prioritising a short duration of the route [3]

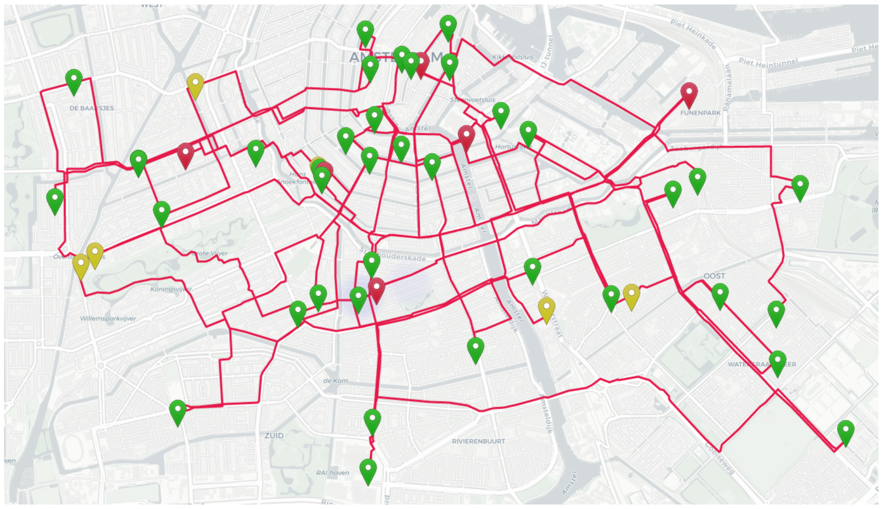


Figure 20: 5000 generations of the GA prioritising on-time deliveries

6.6 Effect Of Post Processing

In some cases, like the route shown in Figure 21 it might be beneficial to run the post-processing to allow the deliverer to wait, because otherwise all deliveries are hours too early. In such a case the post-processing step can allow the deliver to wait to improve the time windows. The result of the post-processing step can be seen in Figure 22. The results do not seem to be that much better, but it should be noted that the post-processing step minimizes the total time window deviation. This means that an order which is 5 minutes too late is favoured over an order that is an hour too early.

[3]Note the colour of the points: yellow means too early, green means on time and red means too late)

Because Figure 22 has more deliveries which are on time, the post-processing step successfully decreased the total time deviation.

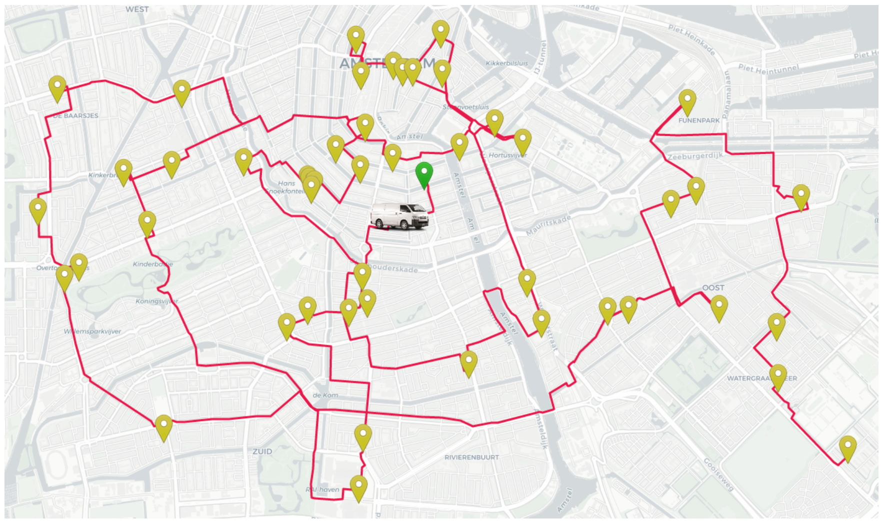


Figure 21: 5000 generations of the GA on a dataset where all deliveries are around 4 hours too early[4]

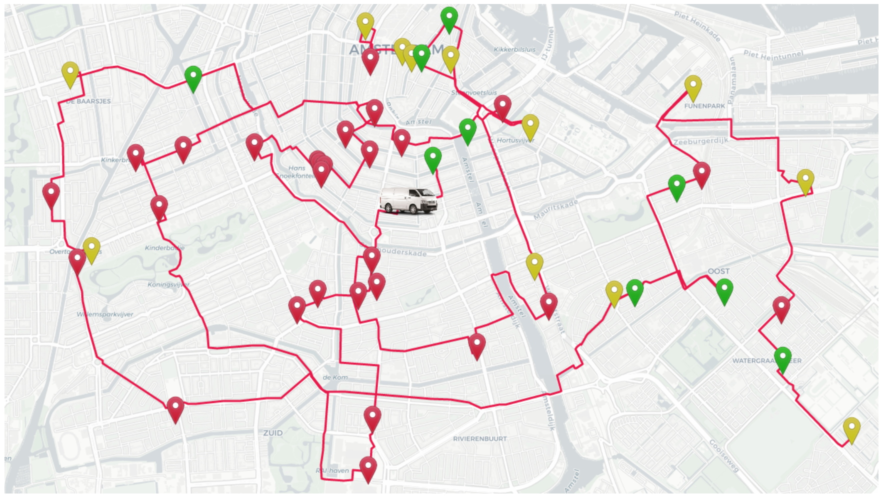


Figure 22: An example of the result in Figure 21 after running the post processing, more deliveries are on time

[4]Note the colour of the points: yellow means too early, green means on time and red means too late)

7 Conclusions

This section presents some main conclusions about the project as a whole. In the first subsection general conclusions about the final implemented system are drawn, in the second subsection more specific conclusions about the performance of the genetic algorithm are drawn from the results as described in section 6.

7.1 Deliverable

At the end of the project the team delivered a solution that included all of the must-haves as described in section 4.1. The system contains at least one algorithm that converges to a (local) optimum in the form of a genetic algorithm. It can adapt to new changes with the greedy insertion algorithms, and a solution is always ready because the system saves the best solution per generation. In addition, A visualiser has been implemented to showcase solutions to the client and the bikes have limited carrying capacity in the algorithm.

The system works well with the existing systems of the client. It is maintainable for a longer period of time through the strongly typed defensive programming style as described in section 3.1 and is extensively tested as described in section 5.6, with statement coverage being over 80%. The processing pipeline makes extending the program with new additions easier. The low coupling of classes combined with the documentation keeps the code maintainable.
The combination of these features create a project complying to service-level-requirements.

7.2 Genetic Algorithm

In this section some general conclusions are drawn from the results presented in section 6. These conclusions are drawn by looking at the bigger picture, instead of just focusing on one particular part of the genetic algorithm, and are made with the final aim of the client in mind.

7.2.1 Parameters

The mutation section shows that exploration is very important for the GA, because with a mutation chance of 0% the solution does not improve past the initial few generations. However, too much exploration can also negatively impact solution performance. Thus, when running the GA a balance between exploration and convergence should be found.

Tournament selection seems to converge faster than roulette selection. However it might be interesting to run the GA for a very long time to see if the roulette selection eventually catches up to the tournament selection, or if it might even be better in the long run. Figure 14 shows that roulette selection explores the solution space much more than tournament selection.

The difference between SinglePoint, TwoPoint and Uniform crossover functions seems to be minimal with no clear pattern emerging. Perhaps some crossover functions work better on certain datasets, this could be explored in the future. It does seem however, that exploration rate and selection function have a much bigger impact on the quality of the solutions.

7.2.2 Clustering

The clustering seems to work correctly. However, due to the nature of K-means and the random initialisation of the cluster centroids, the clusters are not perfectly balanced. It might be beneficial to add some constraint which helps balance out the clusters more evenly across deliverers, like setting a maximum size per cluster.

7.2.3 Capacity

Capacity works as intended. However problem sizes might become too big for one deliverer to efficiently deliver. In such cases it is probably beneficial to increase the amount of clusters (and thus the amount of bikes which deliver and depots to pick up flowers from).

7.2.4 Weights

Weights work correctly, but can be hard to balance. When prioritising duration the route intuitively looks correct. However when prioritizing time windows above all else the GA might create a route which is unnecessarily long to minimise time window violations. An extreme case could be when the GA knows that it will be 3 hours too early; when only prioritising time windows the GA will find a route which takes 3 hours to get to that single client on time. A balance between the two weights should be found. The ideal setting for the weights is dependent on the dataset, and can be found either by trial-and-error or can be optimised using some other algorithm (the latter option is outside of the scope of this project).

7.2.5 Postprocessing

The results show that the postprocessing option works, however it might seem unintuitive that it could result in more clients getting their deliveries late. Instead of trying to minimise the total time window violations it might be more beneficial to prioritise early deliveries over late deliveries. This is dependent on the preferences of the user and can be tweaked in the postprocessing code.

8 Discussion

The following section contains a reflection on the development process and on the delivered product. It discusses the process, the algorithm performance and potential ethical implications. Furthermore it offers a set of future recommendations and reflects on the feedback delivered by the Software Improvement Group (SIG).

8.1 Process

The initial planning contained features that when combined were unachievable in the given time. This was a result of the short research sprint not having the main focus of creating a perfect time planning. The provided support from the client on technical challenges as well as the support from university supervisors on academic challenges created an environment where the team could focus on the tasks of the bachelor project without introducing problems that were outside the scope. Besides the start, the process was comparable to one of a standard software project.

The addition of scrum and the other project management tools helped gain perspective on how projects with larger teams function. An especially good addition to the process was that of the speedboat meetings. They provided a good way to reflect on the current process and on potential improvements. Everyone had to write something down, which means ideas that may not have been mentioned otherwise would still get picked up here.

8.2 Algorithm Performance

Whilst the product is the result of a successful software project, some imperfections still exist. The way the fitness function weighs each part causes the fitness to not be a linear representation of how good the solution is. On top of that the weights do not have very intuitive values. This also results in roulette selection performing worse due to better solutions not having the appropriate higher chance of being selected. This is combined by the performance of the system not being easily checkable.

The final product contains an implementation of fuzzy c-means clustering, however it is not used in the genetic algorithm. Therefore, the impact of c-means on the algorithm performance is not known.

8.3 Ethical Implications

The system that was developed in this project does not have very obvious ethical problems. However, there are some features that can introduce some concern regarding privacy for the customer as well as the deliverer.

8.3.1 Privacy Route Planning

The developed system would allow the indirect tracking of deliverer. In the current implementation this can only be done using the deliverer's schedule. However, a feature that the client might implement later involves the live tracking of deliverers to make the predicted delivery times as accurately as possible. Live tracking of a person's whereabouts could be a privacy issue, however it does not seem very impactful because the location data of the deliverer is no longer useful after the deliveries have been done, which means the data does not need to be stored.

8.3.2 General Data Protection Regulation

The datasets containing the deliveries are held in the database of the backend. This means that if the contents of the database are leaked, the personal information of customers is made public. To prevent this the datasets should be anonymised when they are no longer useful. Keeping the datasets without properly anonymising them or having explicit consent of the customer violates the General Data Protection Regulation [23].

8.4 Future Recommendation

Resulting from some features being cut as mentioned in section 4.3, these features have been re-evaluated in retrospect and this results in the following recommendations for future versions.

8.4.1 Congestion and Feedback

The main feature that proved infeasible in our time window was the implementation of congestion control. This is a very decoupled feature that should be implemented onto the existing distance tables. By implementing this feature, resulting navigation time would closer represent actual navigation time.

8.4.2 Multiple Instances

The project contains an option to deploy the server as a docker image. Working more on this feature can extend the system to run an instance for each calculation. With this, the system can run calculations in parallel instead of sequential, improving the scalability of the project and reducing the negative effects of running multiple algorithms.

8.4.3 Academic Spin-off

Improvements can be made on the extensibility of the algorithms options and related functions. The project contains a good amount of test datasets that are easy to extend. If more algorithms implemented on the backend, it could be used in combination with the visualiser for algorithm testing. However, this was not part of the assignment. The algorithm testing is part of the backend though it is disabled on production builds.

8.4.4 Order Types

The current system simply saves the carrying capacity of a bike as a number. This means there is no distinction with respect to what is actually ordered. However, this would be a nice feature to have if the system is to be deployed in other areas where the carrying capacity changes greatly depending on what is ordered. Therefore a future improvement would be to implement this.

8.4.5 Algorithm Types

During the research phase multiple types of algorithms were found that could be used in this project. The genetic algorithm was chosen, however it would make the system more robust if other types of algorithms were implemented as well. This could then be the basis for a more in-depth study on the optimal way to solve the problem tackled in this project.

8.5 Software Improvement Group

During the development process there were two times where the source code was sent to the SIG for an assessment. The SIG's assessment contains an analysis of the software quality on many aspects, including maintainability, reliability, scalability and security. Their feedback helped prevent delays on this project by early identification of issues. For the first email see appendix B.

8.5.1 Feedback

SIGs feedback on the first deliverable can be summarized in the following points:

- Lower Unit complexity; some functions are too complex. They should be split into multiple simpler functions and must be named accordingly
- Lower Unit size; functions should not be longer than needed. They should be split into multiple smaller functions.
- The project structure complexity of the project was too high. Initially they could not find any tests, while the folder was called test and a readme was delivered in which the project was described (including tests) as well.

8.5.2 Improvement

From the SIG feedback the following alterations were made respectively

- Complex functions were split into multiple simpler functions, as instructed, whilst keeping a test coverage of over 80%.
- Long functions that executed a streak of sequential algorithm tasks were shifted to a Runner class that made the process easier to monitor, and reduced function length.
- Extra indicators of project structure were added to make sure future assessment professionals would waste no time searching.

References

[1] "an open-source javascript library for interactive maps." [Online]. Available: https://leafletjs.com/

[2] "The best apis are built with swagger tools." [Online]. Available: https://swagger.io/

[3] "Electron — build cross platform desktop apps with javascript, html, and css." [Online]. Available: https://electronjs.org/

[4] "Jenkins - build great things at any scale." [Online]. Available: https://jenkins.io/

[5] "Jest - delightful javascript testing." [Online]. Available: https://facebook.github.io/jest/

[6] "Maketek - a delftware company." [Online]. Available: https://maketek.nl/

[7] "Osrm testsuite." [Online]. Available: https://github.com/Project-OSRM/osrm-backend/blob/master/docs/testing.md

[8] "Phacility - phabricator." [Online]. Available: https://www.phacility.com/phabricator/

[9] "Project osrm." [Online]. Available: http://project-osrm.org/

[10] "Redis." [Online]. Available: https://redis.io/

[11] "Tslint - an extensible linter for the typescript language." [Online]. Available: https://palantir.github.io/tslint/

[12] "Typeorm - amazing orm for typescript and javascript (es7, es6, es5). supports mysql, postgresql, mariadb, sqlite, ms sql server, oracle, websql databases. works in nodejs, browser, ionic, cordova and electron platforms." [Online]. Available: http://typeorm.io/

[13] "Typescript - javascript that scales." [Online]. Available: https://www.typescriptlang.org/

[14] "What is agile software development? — agile alliance." [Online]. Available: https://www.agilealliance.org/agile101/

[15] "Dsdm atern handbook (2008)," Jun 2017. [Online]. Available: https://www.agilebusiness.org/content/moscow-prioritisation-0

[16] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2-3, p. 191203, 1984.

[17] S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic algorithms and traveling salesman problems," *European Journal of Operational Research*, vol. 93, no. 3, p. 490510, 1996.

[18] D. Edlich, "List of nosql databases [currently ¿225]." [Online]. Available: http://nosql-database.org/

[19] R. A. Fisher, *Statistical tables for biological, agricultural and medical research.* Longman, 1990.

[20] L. Grandinetti, F. Guerriero, F. Pezzella, and O. Pisacane, "The multi-objective multi-vehicle pickup and delivery problem with time windows," *Procedia - Social and Behavioral Sciences*, vol. 111, pp. 203–212, feb 2014. [Online]. Available: https://doi.org/10.1016/j.sbspro.2014.01.053

[21] L. v. Horen, “Food & agri home,” Dec 2017. [Online]. Available: https://research.rabobank.com/far/en/sectors/regional-food-agri/flourishing_flowers_promising_plants_changes_in_consumer_behaviour.html

[22] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992

[23] C. of the European Union”, “Council regulation (eu) no 679/2016,” 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32016R0679

[24] G. J. Razali, N.M., “Genetic algorithm performance with different selection strategies in solving tsp,” in *Proceedings of the World Congress on Engineering 2011 Vol II*, 2011, pp. 417–431.

[25] Reduxjs, “reduxjs/react-redux,” Jun 2018. [Online]. Available: https://github.com/reduxjs/react-redux

[26] P. Sun, S. Dabia, L. Veelenturf, and T. van Woensel, “The time-dependent profitable pickup and delivery traveling salesman problem with time windows,” Technische Universiteit Eindhoven,” WorkingPaper, 11 2015.

[27] J. Terada, H. Vo, and D. Joslin, “Combining genetic algorithms with squeaky-wheel optimization,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press, 2006. [Online]. Available: https://doi.org/10.1145/1143997.1144203

[28] H. A. Waisanen, D. Shah, and M. A. Dahleh, “A dynamic pickup and delivery problem in mobile networks under information constraints,” *IEEE Transactions on Automatic Control*, vol. 53, no. 6, pp. 1419–1433, jul 2008. [Online]. Available: https://doi.org/10.1109/tac.2008.925849

A Preliminary research report

Algorithmically solving scheduling and routing for flower delivery

The TD-MO-PD-SPDP-MPP-TW

Preliminary Research Report

Ilja Bakx, Jelle Eysbach, Chris Mostert, Casper Schröder

1 Problem description

According to a recent study by Rabobank, flower and potted plant expenditures are expected to increase by 2% per year in Europe and North America. With millennials being likely to buy more plants online, the sales are expected to keep growing over the coming years [15].

Thus, when running an online flower shop in the capital of the Netherlands, it is important to be able to keep up with the growing demand. To be able to do this, it's important to be able to efficiently plan a route so that all customers can be served on time.

The problem of planning this route and the corresponding schedule is similar to the Pickup and Delivery Problem, except instead of each transportation request specifying a single origin and a single destination and all vehicles depart from and return to a central depot, we have several types of requests, several destinations and several (moving) pickup points. It is also extended by adding time windows and last minute changes, making it a partially dynamic problem. We have called our problem the TD-MO-PD-SPDP-MPP-TW: Time Dependent Multi Objective Partially Dynamic Selective Pickup and Delivery Problem with Movable Pickup Points and Time Windows. We chose this name because:

- (TD) The problem is Time Dependent, because the speed of the vehicles will differ throughout the day.
- (MO) The problem is Multi Objective, because we want to optimise the routes of the vehicles and the positions of the pickup points.
- (PD) The problem is Partially Dynamic, because deliveries can change or be added after the main algorithm is done running. In this case a different part of the algorithm will insert this order into one of the routes in a greedy way. The routes won't be completely changed, because of the expected runtime of the algorithm and because the people doing the deliveries want to have reliable schedules.
- (S) The problem is selective, because the vehicles don't have to visit every pickup point.
- (MPP) The problem is extended by the pickup points being able to move around.
- (TW) The problem is extended by having time windows for every delivery point.

The following features are requested by the client:

- Support for multiple bikes.

- Updated delivery time estimates.
- The navigation algorithm will use open source traffic data to take the least congested route.
- Creating clusters of deliveries to be picked up by one bike.
- Scheduling Refill moments at edge-of-city hubs.

The end goal of the client is to create an open platform where Amsterdam based companies can work together to combine deliveries and reduce their time on the road, and consequently reduce their impact on traffic density and pollution and increase overall road safety within the city.

Perhaps interesting to note is that with the new GDPR the rules and regulations regarding data collection are stricter, which means tracking real-time positions of deliverers is legally more difficult. This means that some other way of accurate delivery estimates might have to be used.

Solving this problem will enable the companies of the platform to supply additional options, tighter time windows and punctual same day delivery. The companies can use their deliverers more efficiently, and the deliverers themselves will be able to properly and more easily do their jobs by following the route given instead of making it themselves.

2 State of the Art

In the following section we will give a brief explanation of each algorithm we came across in our research. In each subsection the algorithm is explained, and example problems and applications from literature are discussed.

2.1 Data Preparation

Before the algorithm can be implemented and used we have to think about how to handle the data we are going to use. There exist some open source solutions that we could use in this project. These are given in this section. A way to generate problem instances for testing is also mentioned.

2.1.1 Multi agent system for simulation

Fikar et al. (2017) use a multi agent system to simulate problem instances. Every object in the problem is modeled as an agent. The behaviour of the bikes, hubs and customers etc. are modeled like this. This allows testing of the algorithm with a lot of different situations that it could realistically encounter in real life [7].

2.1.2 Clustering

We can use clustering as a way to find the positions of the hubs. These positions can then be used by the algorithm to plan the routes of the bikes. For the clustering k-means, as defined by MacQueen (1967), with k being the amount of available hubs can be used [23].

Using regular k-means might give sub-optimal clusterings for nodes that are near the edge of the clustering. Therefore it might also be a good idea to try overlapping k-means, as mentioned by Khanmohammadi et al. (2017), to cluster the delivery points and then schedule them with the rest of the algorithm [19].

Mohd et al. (2012) define a new kind of k-means algorithm called max-d k-means. It does not require the user to define k. Max-d k-means has also been shown to be a significant improvement over k-means by the authors. The main advantage of using max-d k-means is that our solution should make a minimal amount of clusterings, in which case it makes more sense to have k be determined by the algorithm [25].

Schuijbroek et al. (2017) use an approach called "cluster-first route-second". In this approach to the vehicle routing problem the delivery points are first distributed over the different vehicles. The objective of the clustering function is to minimise the routing costs. Calculating the exact routing costs of a clustering is too computationally expensive so heuristics have to be used. The routing distance is approximated with a Maximum Spanning Star Approximation. However, the approach used by Schuijbroek et al. does not scale well as the final clustering algorithm essentially explores the whole solution space [30].

2.1.3 Open street map / GraphHopper

Open street map is an open source collaborative map that we can use for routing and visualisation. We can use GraphHopper to determine the cost of a route [2][3].

2.1.4 Leaflet

Leaflet is a Javascript library for interactive maps. We can use this for visualization of a problem and the resulting scheduling and routes [1].

2.2 Main Algorithm

This section will describe several different approaches found in literature for tackling the problem of planning a cost efficient route when dealing with customers and deliveries, as well as the scheduling of the orders.

2.2.1 Genetic Algorithm

A Genetic Algorithm (GA) is an algorithm that is based on the idea of evolution; more concisely, the idea of a set of solutions breeding to create different / better solution [12]. Parts of the solutions are combined to create a new generation, which are evaluated on a certain fitness function. Some solutions are mutated, meaning a part of the solution is randomised, to prevent getting stuck in local optima. This process of making a new generation is repeated until certain criteria are met.

In their paper, Jih et al. (2010) use a variant of a GA, namely the Family Competition Genetic Algorithm (FCGA) to solve the Pickup and Delivery Problem with Time Windows (PDPTW) [17]. As they state, Genetic Algorithms are known for their effectiveness in global optimisation by being able to quickly explore the solution space. The algorithms have been used to solve the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). Because of their strength in routing related problems, they chose to use an adapted version to solve the PDPTW. They stress the importance of the fitness function and the constraints within. Their fitness function results in the travel cost with a certain penalty based on the constraints with the aim of minimizing this value.

How the construction of the next generation is done is crucial to the algorithm. Ghoseiri and Ghannadpour (2009) explain that classical crossover is not appropriate for the TSP or the VRP, due to duplication or omission of vertices [9]. Instead they propose two other ways to handle crossover. Heuristic crossover, which swaps parts of the parents to avoid duplication, is used to produce the first. This way only one child is produced from two parents. To produce another, Merge crossover is used. This process chooses one node at random, and follows with the ones whose time window comes earlier. This way only feasible solutions are produced.

When using a GA for Pickup and Delivery Problems (PDP), solutions are often not feasible because they do not satisfy the constraints. In their paper, Liao and Ting (2010) propose to use this type of algorithm to resolve the Selective Pickup and Delivery Problem (SPDP) [22]. They propose to use a repair strategy to deal with the infeasible offspring. To handle solutions that contain vehicles with negative load, they use a minimum spanning tree to add pickup nodes. To repair inappropriate visiting orders, they insert the starting pickup point between the node with the lowest load and its successor.

2.2.2 Simulated Annealing

Simulated annealing is based on the metallurgy technique called annealing. In this technique metal is heated and gradually cooled which increases the strength of the material.

Simulated annealing, as explained by Kirkpatrick et al. (1983), explores the search space starting at a high temperature. At high temperature the exploration of the search space is less dependent on the cost function; the algorithm will sometimes move to a worse solution due to the temperature. As the temperature gradually lowers the algorithm will tend to solutions that are better than the current iteration as it essentially turns into a gradient descent. This causes it to explore the search space enough at low temperatures to not get stuck in a local optimum and converge when the temperature is low. [20]

2.2.3 Min/Max Ant Colony Optimisation

Chang et al. (2016) propose to use Min/Max Ant System (MMAS), a variant of Ant Colony Optimisation, to solve the SPDP [5]. In this algorithm, ants are constructed that move to other nodes with a probability based on the pheromones of the adjacent nodes. These pheromones evaporate over time, and are increased when an ant that crossed the node reaches its destination. The value of the deposited pheromone is based on the best ant of the iteration, or the global best. The MMAS introduces a minimum and maximum value of these pheromones to prevent premature convergence.

2.2.4 Squeaky Wheel Optimisation

Squeaky Wheel Optimisation is a general approach to optimising, which is described by Jolsin et al. (1999) as a greedy algorithm which is used to construct a solution and is then analysed to find trouble spots. The trouble spots can be improved to gain a large increase in the objective function score. They also show that SWO can be used for scheduling problems [18].

SWO has also been extended to include an evolutionary element. This variant prevents SWO from doing a full construction of a solution from an empty assignment at the start of each iteration, instead keeping the good components and only using SWO to reconstruct the worse parts of the solution [21].

Terada et al. (2006) combine the SWO approach with Genetic algorithms. They show that combining ideas from SWO with a standard GA yields significant improvements over both [33].

2.2.5 Tabu Search

Tabu search is a search strategy for solving optimisation problems designed by Fred Glover in 1988. The problem looks at immediate neighbours to a solution to check for a better variation of this solution. However, normally this local neighbourhood search can get stuck in a local minimum easily. Tabu search marks previously visited solutions as 'taboo' to prevent the algorithm from going back and getting stuck. In addition, the algorithm also allows to take a move which makes the solution worse, in the hopes of finding a better optimum. [10][11]

Potvin et al. (1996) used the Tabu search heuristic to solve the Vehicle Routing Problem with Time Windows in an effective way [27]. Another approach by Gendreau et al. (1994) uses the Tabu Search Heuristic to solve the vehicle routing problem with capacity and route length restrictions [8].

Other implementations by Cordeau et al. (2007) use tabu search as one of several heuristics to solve the One-to-one Pickup and Delivery problem [6], and the implementation by Han et al. (2017) combines the tabu search heuristic with dynamic programming to solve the Attended Home delivery (AHD) problem, keeping in mind no-show and random response times in delivering [13].

2.2.6 Hybrid Heuristic for Attended Home Delivery

Attended home delivery is an extension of the last mile problem. A well known problem originating from the telecommunication field, but has since been adapted to the last leg of supply chain management, It refers to the provision of distribution service from a depot to the home of customers, which takes place at the end of a logistics distribution. Important included factors are the inclusion of no-show and random response time, meaning for every customer the time they will take to respond to the delivery is randomised to imitate a real life scenario. This is implemented by combining appointment scheduling and VRP with soft time windows. As a solution a pipeline of three heuristic approaches is created. First a tabu search to minimize the routing model, secondly, with the distribution order for each vehicle a heuristic dynamic programming is used to created approximate cost for the appointment scheduling model. Lastly, both are combined into a hybrid heuristic algorithm to solve the integrative model. [13]

2.2.7 Large/Adaptive Neighbourhood Search

(Large) Neighbourhood search is a search strategy proposed by Shaw in 1998 [32]. It is a search strategy which looks at neighbouring solutions to see if they are better and then modifies the current solution incrementally. The class of very large-scale neighbourhood search algorithms have an exponentially sized neighbourhood [26]. Another variation, proposed by Mladenović and Hansen (1997), is the variable neighbourhood search, which allows the neighbourhood structure to vary during runtime [24].

Reyes et al. (2015) use the neighbourhood search as an improvement heuristic to solve the Vehicle routing problem with roaming delivery locations [28], while Cordeau et al. (2008) use a variable neighbourhood search to solve the Single Vehicle Pickup and Delivery Problem with Last in first out constraints [6].

The Large Neighbourhood search uses various heuristics to add or remove parts of the solution. Ropke and Pisinger (2006), for example, use different heuristics with a roulette wheel selection. Which heuristic is used depends on the performance of that heuristic so far. The performance of a heuristic is determined by a set of three metrics. The first metric is how many times the heuristic resulted in a new global optimum. The second is how many times the heuristic created a solution that had has lower cost than the current solution. The final metric is how many times the solution creates a solution that has not been accepted before but has a higher cost than the current solution. These metrics cause both convergence and exploration of the search space. [29]

2.3 Greedy Pickup

When a new order comes in while a solution already exists, the solution has to be updated. The most efficient way to achieve this is to use a heuristic to add the new order to the current solution and then re-run the optimisation algorithm to improve it further. This section mentions some of the heuristics that can be used for this.

2.3.1 Basic greedy Insertion Heuristic

Ropke and Pisinger (2006) and Jiajia et al. (2017) use a greedy insertion heuristic. It heuristically inserts every new order starting with the order the furthest away from the distribution center. It then checks every position in the current schedule to see where the new order can be inserted. This insertion keeps in mind that the time windows of the deliveries should not be violated. If an insertion resulting in a feasible solution is possible the cost of the insertion is determined, otherwise the cost is some maximum value. Then the algorithm finds the best position to insert the order, i.e. the one with the lowest cost. It then checks if all new orders are inserted into feasible solutions. If this is not the case the leftover orders are moved to the next scheduling period or a new vehicle is used. [16][29]

2.3.2 Regret Heuristic

An algorithm using the Regret Heuristic can reconsider decisions made in past steps. It can reverse decisions if their cost becomes considerable. Hassin and Keinan (2008) use this idea to improve a greedy algorithm for the TSP [14]. Ropke and Pissinger also use it as an insertion heuristic in their Large Neighbourhood Search to solve PDP with Time Windows [29].

2.3.3 Shaw Removal Heuristic

An iterated solution procedure was proposed in which large neighborhood search is used for distribution planning. First introduced by Shaw (1997), the Shaw removal heuristic removes requests that are closely related [31]. This can be in terms of distance, start time of service, size of the load and vehicles that are able to serve these requests. These requests have a larger probability of being removed from the solution [34]. The concept behind removing similar requests is that there can be more more possibilities when reinserting the requests into the solution. A request is more likely to be inserted in routes that previously contained related requests.

2.4 Competitors

There are companies which handle similar problems. Graphhopper for example, however this service is expensive and handles the Traveling Salesman Problem instead of the TD-MO-PD-SPDP-MPP-TW. There is no company that handles this problem, and the ones that handle subproblems do so in a limited manner or for a fee. This is why this project is valuable from a commercial point of view.

3 Requirements

A prioritisation of all considered features is given in section 4. In this section the strict requirements as agreed on by the client are summarised. A distinction between functional and non-functional requirements is made.

3.1 Functional

1. Find routing and scheduling that does not violate time windows
   The main goal of the algorithm is to find a solution of the TD-MO-PD-SPDP-MPP-TW. The solution must be close to optimal and not violate any constraints.

2. Make the algorithm dynamic
   The algorithm must be able to adjust to new changes. New orders could be received at a time where a good solution has already been found. The algorithm must be able to integrate these into the current solution without too many radical changes in the solution.

3. Anytime algorithm
   The algorithm should be able to give a result at anytime, so that it can output a result when needed. This will ensure that there will always be a route, even when the algorithm is not "finished".

4. Make the algorithm work for real life data / data preparation using a dummy dataset
   The algorithm must be tested with a dummy dataset that is representative of real life data. If it is possible it must also be tested with real life data.

5. Updated delivery time estimates
   The algorithm must be able to accurately show delivery time estimates to the customer while it is running.

6. Each customer is served only once per order.
   This requirement ensures that customers are not visited twice unnecessarily.

7. Bikes have a carrying capacity that can not be exceeded.
   It is important that bikes do not run out of goods before arriving at a customer.

8. Customers have to be served within their time window if possible. If not possible they have to be notified and given their updated time window.
   The primary use case of our solution is the delivery of flowers. In this use case time windows are relatively flexible and allow solutions to not adhere to all time windows exactly if it creates a much better solution.

9. It must be possible to run a simulation on dummy data to test the algorithm.
   The algorithm should be tested on some datasets, for this datasets must first be generated.

10. The algorithm must support visualisation to showcase its solutions.
    The algorithm must output solutions in a sensible and usable manner to allow visualisation.

11. The algorithm must be able to connect to the existing back-end and the deliverers.
    Our product should have some kind of API to communicate with external systems so the client's software can connect to it.

3.2 Non-Functional

1. The algorithm must be extensible.

2. The algorithm must be efficient enough to be able to process orders real time.

3. The algorithm must be written in JavaScript using NodeJS.

4. The algorithm must be properly documented.
5. The algorithm must be written in a maintainable way.
6. The API (if applicable) must be documented with Swagger.

4 Feature prioritisation (MoSCoW)

MoSCoW is a technique to prioritise requirements in a software engineering process. It is an acronym of the phrases: ”Must have”, ”Should have”, ”Could have” and ”Won’t have” [4].
This section contains the prioritisation of the features of the product that is being made during this project.

4.1 Must-Haves

- At least one main algorithm which converges towards an optimal solution using algorithms described in this report.
- Make the algorithm dynamic, i.e. it should be able to quickly adapt to changes
- The algorithm has to be an anytime algorithm
- Data preparation using a dummy dataset
- Make the algorithm work for real life data
- Updated delivery time estimates
- Algorithm must be able to connect to existing back-end and the deliverers

4.2 Should-Haves

- Visualise solution in an existing application / Create an own visualisation (heat map)
- ”Live” arrival estimation times per delivery point (showcase of heuristics)
- Take capacity usage into account

4.3 Could-Haves

- Congestion Control
- Allow for dynamic pickup spots
- Use open source traffic data to take the least congested route
- Hyper-heuristics
- Be able to handle last minute changes (like traffic jams or broken bridges)

4.4 Won’t Haves

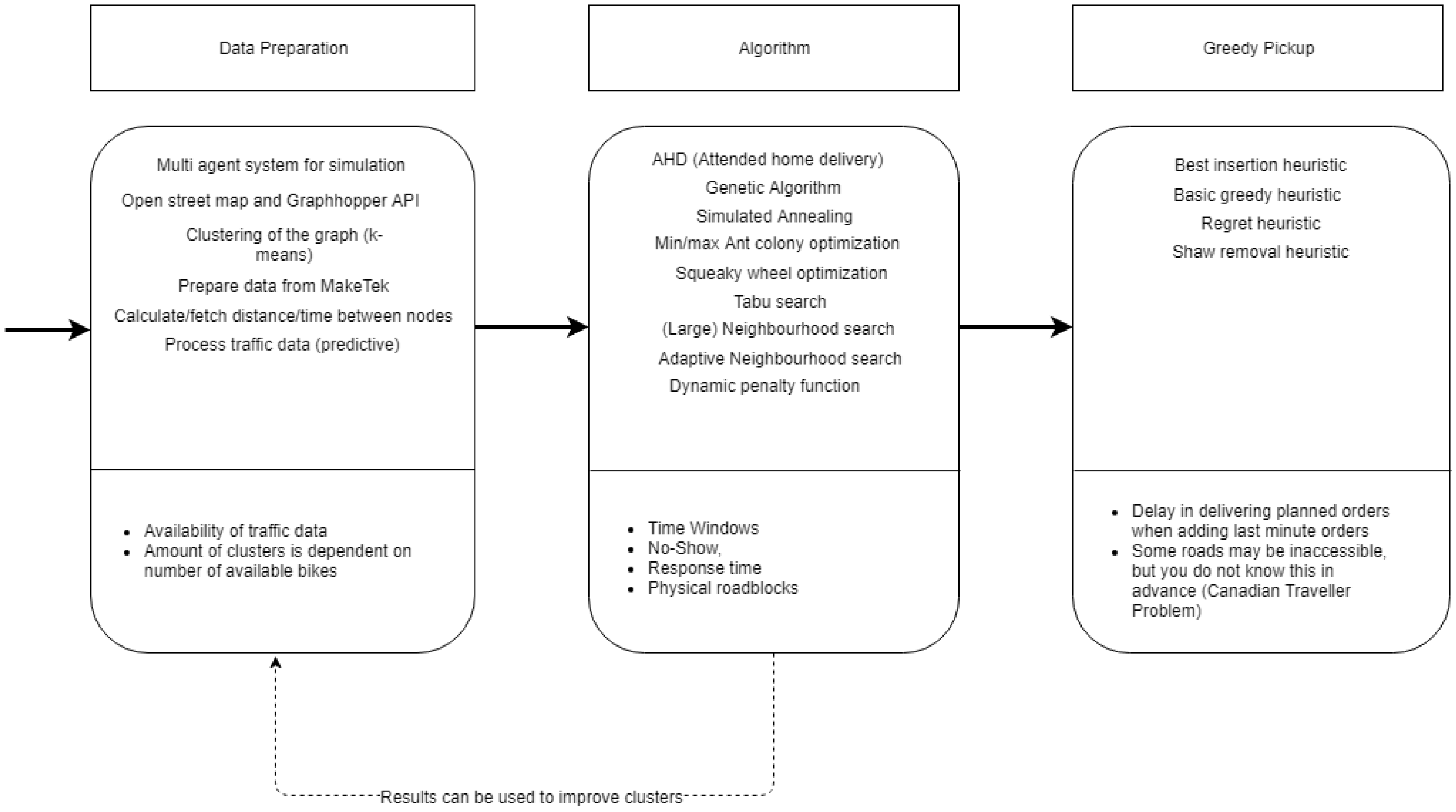
We have not encountered features that we will not consider.

4.5 Feasibility analysis

All the features that are mentioned in this report seem feasible. The features have already been inserted in the project planning. The proposed pipeline of algorithms provides a stable base for the solution, and can be shrinked or extended if time constraints would endanger the outcome.

5 Proposal

This section explains how our project will be set up, including data preparation and what tools and algorithms we will use. We will use NodeJS for the server side code and make our visualisations in web pages to be displayed in a browser. We will divide the work over our group with tickets and we will use Phabricator to keep track of these tickets. Version control will be done with git. We will use Mocha as testing framework for our NodeJS code.



Data Preparation
Algorithm
Greedy Pickup
Multi agent system for simulation
Open street map and Graphhopper API
Clustering of the graph (k-means)
Prepare data from MakeTek
Calculate/fetch distance/time between nodes
Process traffic data (predictive)
Availability of traffic data
Amount of clusters is dependent on number of available bikes
AHD (Attended home delivery)
Genetic Algorithm
Simulated Annealing
Min/max Ant colony optimization
Squeaky wheel optimization
Tabu search
(Large) Neighbourhood search
Adaptive Neighbourhood search
Dynamic penalty function
Time Windows
No-Show,
Response time
Physical roadblocks
Best insertion heuristic
Basic greedy heuristic
Regret heuristic
Shaw removal heuristic
Delay in delivering planned orders when adding last minute orders
Some roads may be inaccessible, but you do not know this in advance (Canadian Traveller Problem)
Results can be used to improve clusters

Figure 1: Algorithm pipeline

In figure 1 the basic proposed pipeline of the project is shown, with the arrows representing the flow of data.

5.1 Data preparation

This subsection will explain how the data is preprocessed before it can be used in the algorithm.

5.1.1 Transitioning from location to a graph

At start a list of delivery nodes is provided, containing : Longitude, Latitude, Product, time window and status. Through usage of a navigation algorithm, per time window, the weighted travel distances between all nodes are calculated and stored. Resulting in a Set with space complexity of $O(tw * n^2)$.

5.1.2 Generating Datasets

To prove that the proposed algorithm works well for most cases, different scenarios are converted into a set of dummy datasets. These sets differ in distribution of nodes, for example close combinations or big outliers. These are used test efficiency of our results for multiple problems. And preventing a bias in the process of choosing the best algorithms.

5.1.3 Calculating distances by bike

To calculate the effective distance between deliveries by bike, multiple resources are already available. For example the GraphHopper and google maps API. As bicycle navigation is still in infancy stage, no clear best resource exists. Both will be implemented to allow the client operating the algorithm to choose and experiment.

5.1.4 Visualiser

To be able to make adjustments to what algorithms are used, the result set must be visualised in a form that is interpretable by humans. This is done in form of a Map of the results, with export to svg for documentation, maybe with json result set. So a database of results can be generated.

5.2 Algorithm Plug-ins

As the goal is to create plug-and-play algorithms, that can be chained or executed concurrently, a single specification per pipeline is used for each of its entries. Consisting of an algorithm object class and a configuration file. Different parts of the pipeline have contract based input and output, guaranteeing compatibility between nodes.

5.3 Main algorithm

The main algorithm will have a genetic algorithm as its basis. This type of algorithm has been proven to be among the best for this type of problem, and is relatively easy to modify in order to handle the specific constraints. Furthermore, a general GA is probably not very difficult to implement.

Depending on the performance of the first implementation of GA on our problem, both in the quality of the solutions and the time the algorithm takes to converge, we can look at other heuristics to combine with the GA. These changes can be in the GA itself, or 'chained', i.e. we use the output of the GA after several iterations as the input for SWO, like Terada et al. (2006) did in their implementation which was shown to improve the quality of solutions over just the GA [33].

It is quite difficult to predict which heuristics work well for our specific problem, and the process will involve some experimentation, especially since we did not find our exact problem in literature due to the specific constraints.

All of the algorithms we encountered in the literature can be seen in the 'algorithm' box of figure 1, however due to the amount of constraints our solution has it might be very difficult to combine ant colony optimisation with the GA approach. The other algorithms allow for the same encoding of the solutions, and the literature described in section 2 has shown that they can all be used for similar PDPs. Thus, we will try combining a GA with the other mentioned algorithms, excluding ant colony optimisation, and empirically check which combinations work well for our problem. Using the roulette wheel approach used by Ropke and Pisinger (2006) to efficiently combine heuristics might also work [29].

5.4 Dynamic order addition

In the case of changes happening after the main algorithm is done running, it may not be optimal to completely run it again. A solution has to be found quickly, so we will run a greedy algorithm, a simulated annealing algorithm, and/or the heuristics described in section 2.3, based on which perform better. This part of the algorithm will be able to handle added and removed orders. Other problems like a customer changing time windows or changing which type of capacity is ordered can be modeled in deletions and additions.

6 Project planning

In this chapter, the project planning is laid out in the form of a sprint planning. This is a rough estimation of what will be done and when.

6.1 Sprint 1 (weeks 1-2)

The first sprint is used to research possible solutions as well as set up a basic workspace and project structure.

- Research report:
  - Problem analysis
  - Existing solutions analysis
  - Feature prioritisation (MoSCoW) and feasability analysis
  - Initial algorithm selection/design
  - Initial set of requirements
- Analysis of requirements and create a Plan of Action (PoA)
- Setup a basic project structure and import necessary modules

6.2 Sprint 2 (weeks 3-4)

The basic data structures will be designed and implemented in this sprint. This includes the data provided by MakeTek and the APIs of grasshopper and leaflet. We will also work on creating a dummy dataset that is as representative of real life data as possible.

- Create basic web application and server for display and demo purposes
- Create first data retrieval method based on chosen platform
- Based on map data create initial data model (in memory)
- Dummy data from provided dummy data source

6.3 Sprint 3 (weeks 5-6)

In sprint 3 well implement a first version of the navigation/scheduling algorithm. It should work with at least one bike. We will also implement dynamic updating of our data structures from the bikes themselves or traffic data.

- First iteration of navigation/scheduling algorithm
- "The lone ranger": a small delivery simulation service (mock)
- Connect to live data source (read only) and listen for changes

6.4 Sprint 4 (weeks 7-8)

We will continue working on the algorithm and add some new features. We will also try extending the pipeline with different algorithms and see how they work. We will also try combining different algorithms and heuristics.

- Second iteration of navigation algorithm
- Optimizing travel routes based on maximum capacity usage
- Start on report

6.5 Sprint 5 (weeks 9-10)

In this sprint well create the final iteration of the algorithm. We will also finish the first draft of the final report.

- Final iteration of navigation algorithm:
- Dummy method to introduce delivery delay (functional ”Live” update mechanic)
- Optimisation using advanced heuristics
- First draft report

6.6 Final sprint (week 11)

In the final sprint, we will finalize the report and prepare the presentation.

- Final draft report
- Setup demo instance and scenario

References

[1] “an open-source javascript library for interactive maps.” [Online]. Available: http://leafletjs.com/

[2] “Graphhopper directions api with route optimization.” [Online]. Available: https://www.graphhopper.com/

[3] “Main page.” [Online]. Available: https://wiki.openstreetmap.org/

[4] “Dsdm atern handbook (2008),” Jun 2017. [Online]. Available: https://www.agilebusiness.org/content/moscow-prioritisation-0

[5] Y.-W. Chang, C.-C. Liao, and C.-K. Ting, “An ant system for the selective pickup and delivery problem,” in *Computer Symposium (ICS), 2016 International.* IEEE, 2016, pp. 94–97.

[6] J.-F. Cordeau, G. Laporte, and S. Ropke, “Recent models and algorithms for one-to-one pickup and delivery problems,” in *The vehicle routing problem: latest advances and new challenges.* Springer, 2008, pp. 327–357.

[7] C. Fikar, P. Hirsch, and M. Gronalt, “A decision support system to investigate dynamic last-mile distribution facilitating cargo-bikes,” *International Journal of Logistics Research and Applications*, vol. 21, no. 3, p. 300317, Feb 2017.

[8] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management science*, vol. 40, no. 10, pp. 1276–1290, 1994.

[9] K. Ghoseiri and S. F. Ghannadpour, "Hybrid genetic algorithm for vehicle routing and scheduling problem," *Journal of Applied Sciences*, vol. 9, no. 1, pp. 79–87, jan 2009. [Online]. Available: https://doi.org/10.3923/jas.2009.79.87

[10] F. Glover, "Tabu search—part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, aug 1989. [Online]. Available: https://doi.org/10.1287/ijoc.1.3.190

[11] ——, "Tabu search—part II," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, feb 1990. [Online]. Available: https://doi.org/10.1287/ijoc.2.1.4

[12] D. E. Goldberg and J. H. Holland, *Machine Learning*, vol. 3, no. 2/3, pp. 95–99, 1988. [Online]. Available: https://doi.org/10.1023/a:1022602019183

[13] S. Han, L. Zhao, K. Chen, Z. wei Luo, and D. Mishra, "Appointment scheduling and routing optimization of attended home delivery system with random customer behavior," *European Journal of Operational Research*, vol. 262, no. 3, pp. 966–980, nov 2017. [Online]. Available: https://doi.org/10.1016/j.ejor.2017.03.060

[14] R. Hassin and A. Keinan, "Greedy heuristics with regret, with application to the cheapest insertion algorithm for the TSP," *Operations Research Letters*, vol. 36, no. 2, pp. 243–246, mar 2008. [Online]. Available: https://doi.org/10.1016/j.orl.2007.05.001

[15] L. v. Horen, "Food & agri home," Dec 2017. [Online]. Available: https://research.rabobank.com/far/en/sectors/regional-food-agri/flourishing_flowers_promising_plants_changes_in_consumer_behaviour.html

[16] Z. Jiajia, L. Guorong, G. Zhenyu, and B. Xiaohui, "Delivery vehicle routing problem with simultaneous delivery and pickup in e-commerce environment," *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017.

[17] W. Jih, C.-Y. Kao, and J. Y. jen Hsu, "Using family competition genetic algorithm in pickup and delivery problem with time window constraints," in *Proceedings of the IEEE Internatinal Symposium on Intelligent Control.* IEEE, 2010. [Online]. Available: https://doi.org/10.1109/isic.2002.1157813

[18] D. E. Joslin and D. P. Clements, "Squeaky wheel optimization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 353–373, 1999.

[19] S. Khanmohammadi, N. Adibeig, and S. Shanehbandy, "An improved overlapping k-means clustering method for medical applications," *Expert Systems with Applications*, vol. 67, p. 1218, 2017.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, p. 671680, 1983.

[21] J. Li, A. J. Parkes, and E. K. Burke, "Evolutionary squeaky wheel optimization: A new framework for analysis," *Evolutionary Computation*, vol. 19, no. 3, pp. 405–428, sep 2011. [Online]. Available: https://doi.org/10.1162/evco_a_00033

[22] X.-L. Liao and C.-K. Ting, “An evolutionary approach for the selective pickup and delivery problem,” in *IEEE Congress on Evolutionary Computation.* IEEE, jul 2010. [Online]. Available: https://doi.org/10.1109/cec.2010.5586360

[23] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics.* Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992

[24] N. Mladenović and P. Hansen, “Variable neighborhood search,” *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, nov 1997. [Online]. Available: https://doi.org/10.1016/s0305-0548(97)00031-2

[25] W. M. W. Mohd, A. H. Beg, T. Herawan, and K. F. Rabbi, “Maxd k-means: A clustering algorithm for auto-generation of centroids and distance of data points in clusters,” *Communications in Computer and Information Science Computational Intelligence and Intelligent Systems*, p. 192199, 2012.

[26] D. Pisinger and S. Ropke, “Large neighborhood search,” in *Handbook of metaheuristics.* Springer, 2010, pp. 399–419.

[27] J.-Y. Potvin, T. Kervahut, B.-L. Garcia, and J.-M. Rousseau, “The vehicle routing problem with time windows part i: tabu search,” *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 158–164, 1996.

[28] D. Reyes, M. Savelsbergh, and A. Toriello, “Vehicle routing with roaming delivery locations,” *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 71–91, 2017.

[29] S. Ropke and D. Pisinger, “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,” *Transportation Science*, vol. 40, no. 4, p. 455472, 2006.

[30] J. Schuijbroek, R. Hampshire, and W.-J. V. Hoeve, “Inventory rebalancing and vehicle routing in bike sharing systems,” *European Journal of Operational Research*, vol. 257, no. 3, p. 9921004, 2017.

[31] P. Shaw, “A new local search algorithm providing high quality solutions to vehicle routing problems,” *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[32] ——, “Using constraint programming and local search methods to solve vehicle routing problems,” in *International conference on principles and practice of constraint programming.* Springer, 1998, pp. 417–431.

[33] J. Terada, H. Vo, and D. Joslin, “Combining genetic algorithms with squeaky-wheel optimization,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation.* ACM Press, 2006. [Online]. Available: https://doi.org/10.1145/1143997.1144203

[34] L. van der Hagen, T. Visser, and R. Spliet, “The pickup and delivery problem with time windows: an adaptive large neighborhood search heuristic,” 2017.

A Glossary

- **(DARP) Dial-A-Ride Problem**

  PDP but for passenger transportation
- **(FCGA) Family Competitive Genetic Algorithm**

  A Genetic Algorithm which adds the element of Families. Families consist of recreations of an individual. Every generation only one member of a family can survive.
- **(GA) Genetic Algorithm**

  An algorithm based on the idea of evolution. Creates a big set of random solutions and keeps breeding these solutions until certain parameters are met. Often used for clustering and automated design.
- **(GDPR) General Data Protection Regulation**

  An EU law on data protection and privacy. Intended to further protect citizens and give them them control over their personal data used by companies.
- **Greedy Algorithm**

  An algorithm which makes a locally optimal choice at each stage of computing to hopefully find a global optimum. Often does not produce an optimal solution, but uses very little time to run.
- **(MMAS) Min/Max Ant Colony Optimization**

  A swarm intelligence based algorithm, using ants (scouts) which leave pheromones based on the length of the route they walked to determine the best route. The Min/Max variant sets a minimum and a maximum limit to the amount of pheromones per part of the route. Often used for solving mazes/maze-like problems.
- **(PDP) Pickup and Delivery Problem** A problem in which each load has to be transported from a depot to a single destination, usually by multiple vehicles.
- **(PDPTW) Pickup and Delivery Problem with Time Windows**

  The PDP with added Time Windows (constraint) for every delivery.
- **(SPDP) Selective Pickup and Delivery Problem**

  The PDP where not all pickup points have to be visited by a vehicle.
- **(SWO) Squeaky Wheel Optimization**

  Uses a greedy algorithm to construct a solution which is then analysed to find trouble spots. When these trouble spots are fixed the general 'fitness' of the solution is increased. This algorithm can fix flaws in imperfect solutions
- **TD-MO-PD-SPDP-MPP-TW**

  Time Dependant Multi Objective Partially Dynamic Selective Pickup and Delivery Problem with Movable Pickup Points and Time Windows.
- **(TSP) Traveling Salesman Problem**

  A problem in which the shortest route through all cities and back to the origin has to be found, given the distances between them.

- **(VRP) Vehicle Routing Problem**

  A problem in which the optimal set of routes for a set of vehicles has to be found so that each customer is visited.

- **(VRPPD) Vehicle Routing Problems with Pickups and Deliveries**

  An extension of the VRP, adding a number of goods that have to be picked up and delivered to the customers.

B SIG feedback emails

Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden in de laatste fase van het project.

Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.

Mochten er nog vragen of opmerkingen zijn dan hoor ik dat graag.

Met vriendelijke groet,

[Feedback]

De code van het systeem scoort 3.7 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Unit Complexity vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.

In jullie geval is geneticAlgorithm() in het bestand met dezelfde naam een goede kandidaat. Jullie geven de verschillende stukken functionaliteit nu met commentaar aan. Door dit in de codestructuur te verwerken, dus met een aparte methode per stap, wordt de code makkelijker te begrijpen. Daarnaast kun je op die manier de stappen apart testen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een descriptieve naam kan elk van de onderdelen apart getest worden en wordt de overall flow van de methode makkelijker te begrijpen.

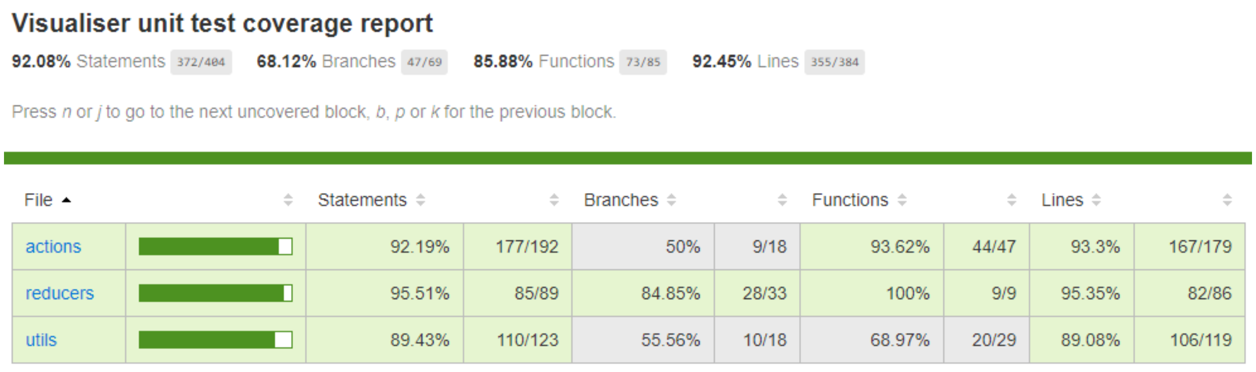
Voor Unit Complexity geldt in jullie geval hetzelde als bij Unit Size: doordat methodes te veel functionaliteit bevatten wordt de methode vanzelf onnodig complex. Door een goed niveau van abstractie te kiezen worden de methodes zowel kleiner als minder complex, dus je kunt beide problemen tegelijk verbeteren.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

```
Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat
    een aantal aanbevelingen die meegenomen kunnen worden in de laatste fase van het
    project.

Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van
     hun code en dient niet gebruikt te worden voor andere doeleinden.

Mochten er nog vragen of opmerkingen zijn dan hoor ik dat graag.

Met vriendelijke groet,


[Hermeting]

In de tweede upload zien we dat het project een stuk groter is geworden. De score voor
    onderhoudbaarheid is in vergelijking met de eerste upload ongeveer gelijk gebleven.

Bij de verbeterpunten uit de eerste upload, Unit Size en Unit Complexity, zien we een
    kleine verbetering. Deze is echter niet genoeg om de totaalscore tot boven de 4
    sterren te laten stijgen.

Zoals jullie per email hebben aangegeven hadden jullie in de eerste upload wel degelijk
    testcode, dus die opmerking uit de feedback op de eerste upload komt te vervallen.
    Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie
    ook nieuwe testcode hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog
    steeds goed uit.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de
    eerste upload deels zijn meegenomen tijdens het ontwikkeltraject.
```

C Visualiser test coverage report



Visualiser unit test coverage report

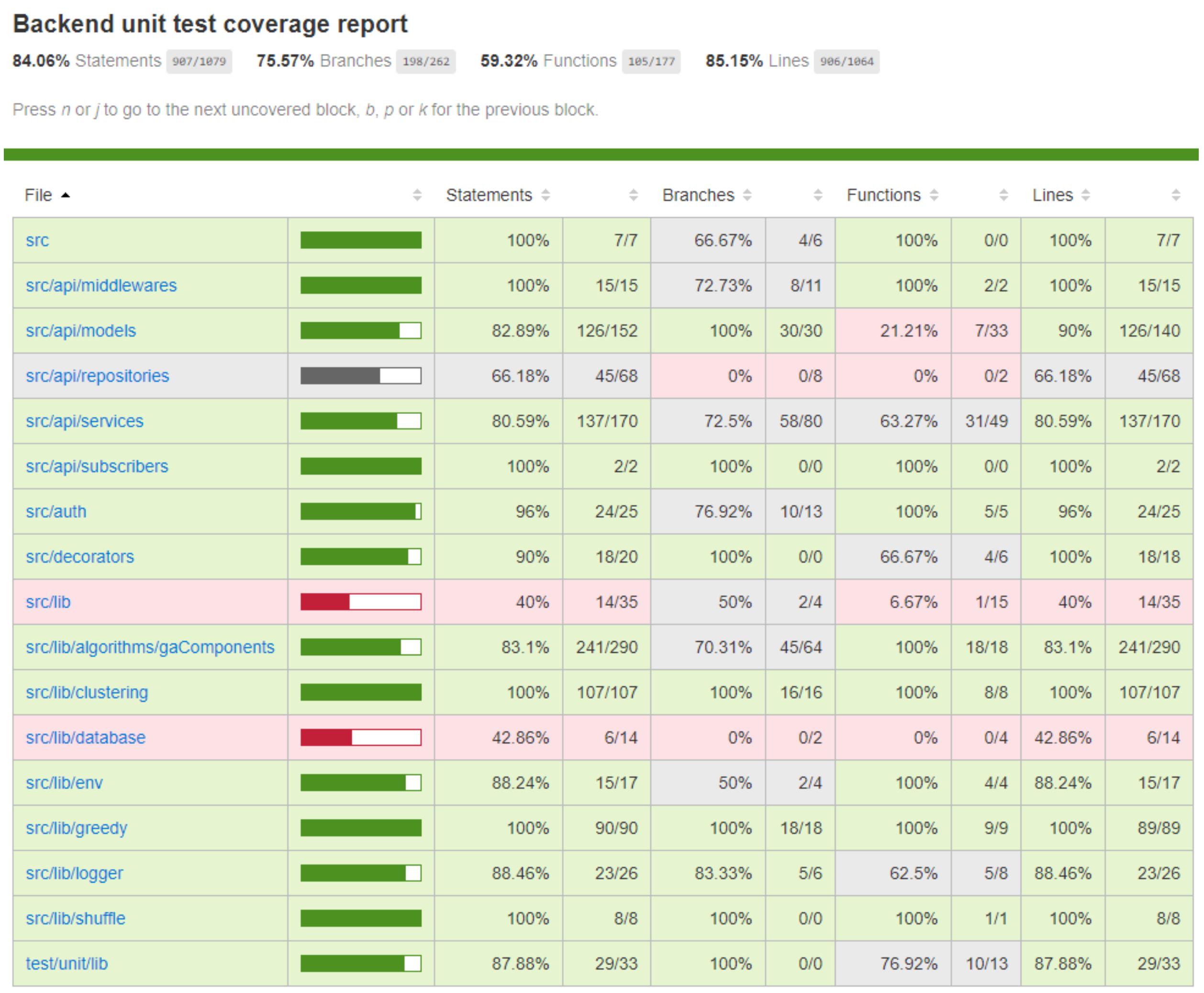
92.08% Statements 372/404 **68.12%** Branches 47/69 **85.88%** Functions 73/85 **92.45%** Lines 355/384

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
actions		92.19%	177/192	50%	9/18	93.62%	44/47	93.3%	167/179
reducers		95.51%	85/89	84.85%	28/33	100%	9/9	95.35%	82/86
utils		89.43%	110/123	55.56%	10/18	68.97%	20/29	89.08%	106/119

Figure 23

D Backend test coverage report



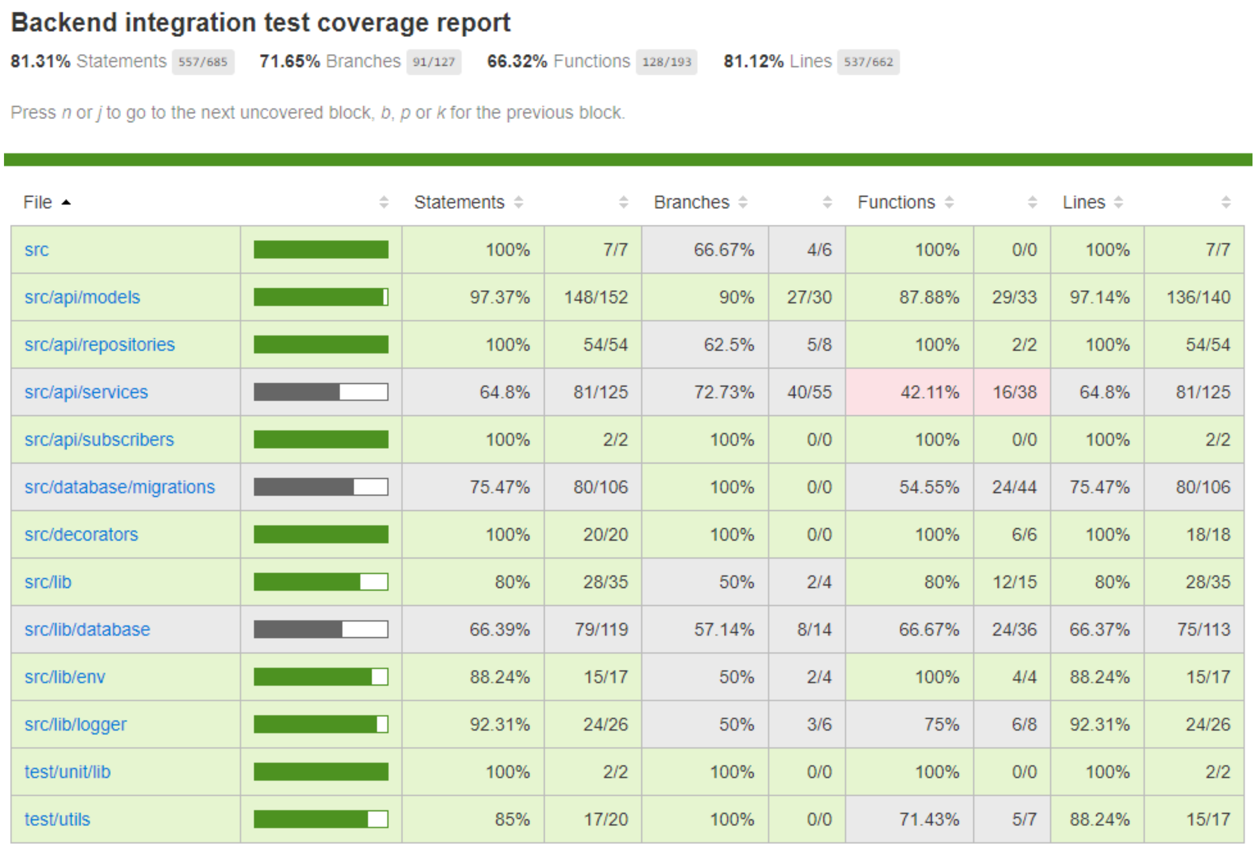
Backend unit test coverage report

84.06% Statements 907/1079 **75.57%** Branches 198/262 **59.32%** Functions 105/177 **85.15%** Lines 906/1064

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		100%	7/7	66.67%	4/6	100%	0/0	100%	7/7
src/api/middlewares		100%	15/15	72.73%	8/11	100%	2/2	100%	15/15
src/api/models		82.89%	126/152	100%	30/30	21.21%	7/33	90%	126/140
src/api/repositories		66.18%	45/68	0%	0/8	0%	0/2	66.18%	45/68
src/api/services		80.59%	137/170	72.5%	58/80	63.27%	31/49	80.59%	137/170
src/api/subscribers		100%	2/2	100%	0/0	100%	0/0	100%	2/2
src/auth		96%	24/25	76.92%	10/13	100%	5/5	96%	24/25
src/decorators		90%	18/20	100%	0/0	66.67%	4/6	100%	18/18
src/lib		40%	14/35	50%	2/4	6.67%	1/15	40%	14/35
src/lib/algorithms/gaComponents		83.1%	241/290	70.31%	45/64	100%	18/18	83.1%	241/290
src/lib/clustering		100%	107/107	100%	16/16	100%	8/8	100%	107/107
src/lib/database		42.86%	6/14	0%	0/2	0%	0/4	42.86%	6/14
src/lib/env		88.24%	15/17	50%	2/4	100%	4/4	88.24%	15/17
src/lib/greedy		100%	90/90	100%	18/18	100%	9/9	100%	89/89
src/lib/logger		88.46%	23/26	83.33%	5/6	62.5%	5/8	88.46%	23/26
src/lib/shuffle		100%	8/8	100%	0/0	100%	1/1	100%	8/8
test/unit/lib		87.88%	29/33	100%	0/0	76.92%	10/13	87.88%	29/33

Figure 24



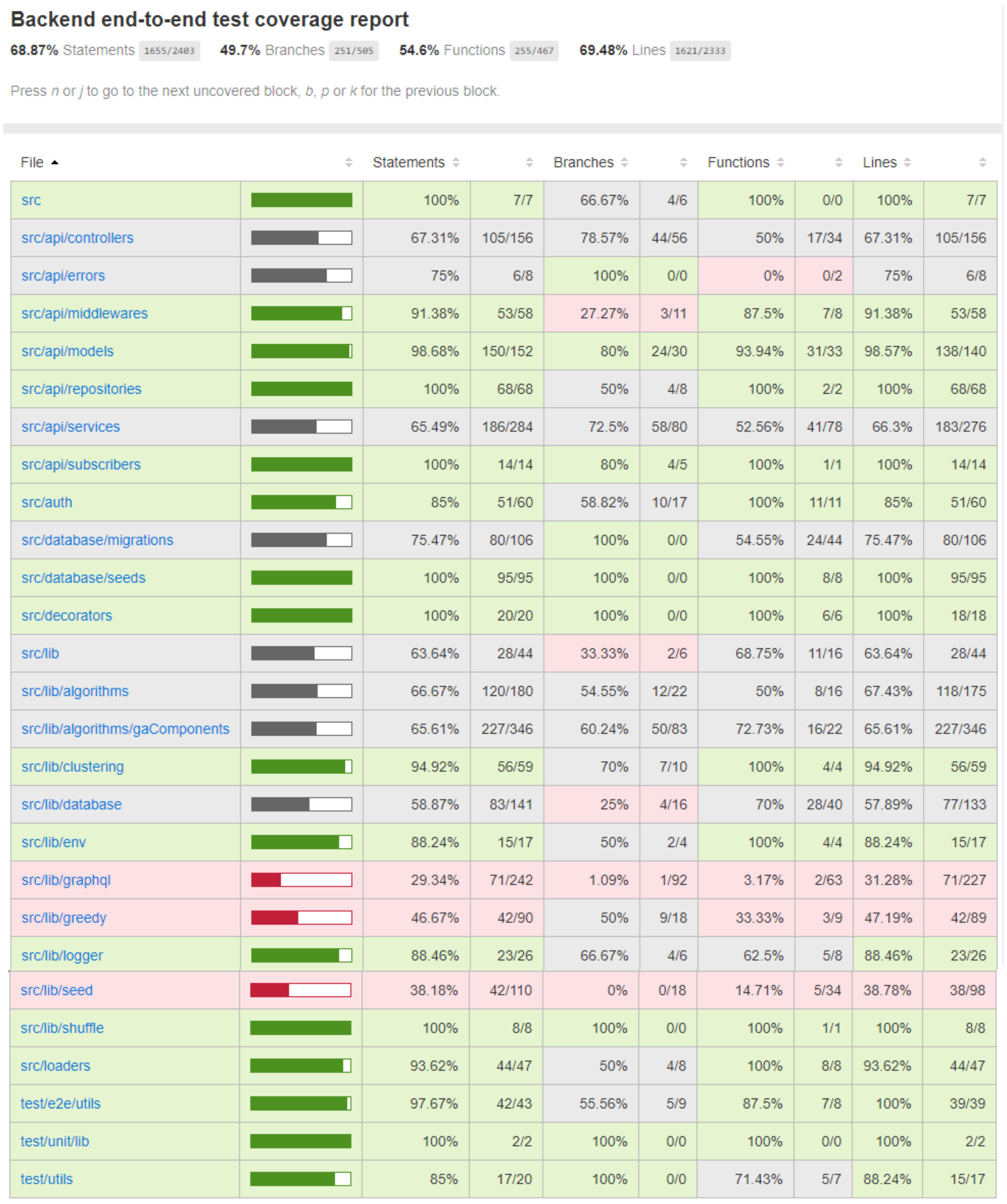
Backend integration test coverage report

81.31% Statements 557/685 **71.65%** Branches 91/127 **66.32%** Functions 128/193 **81.12%** Lines 537/662

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		100%	7/7	66.67%	4/6	100%	0/0	100%	7/7
src/api/models		97.37%	148/152	90%	27/30	87.88%	29/33	97.14%	136/140
src/api/repositories		100%	54/54	62.5%	5/8	100%	2/2	100%	54/54
src/api/services		64.8%	81/125	72.73%	40/55	42.11%	16/38	64.8%	81/125
src/api/subscribers		100%	2/2	100%	0/0	100%	0/0	100%	2/2
src/database/migrations		75.47%	80/106	100%	0/0	54.55%	24/44	75.47%	80/106
src/decorators		100%	20/20	100%	0/0	100%	6/6	100%	18/18
src/lib		80%	28/35	50%	2/4	80%	12/15	80%	28/35
src/lib/database		66.39%	79/119	57.14%	8/14	66.67%	24/36	66.37%	75/113
src/lib/env		88.24%	15/17	50%	2/4	100%	4/4	88.24%	15/17
src/lib/logger		92.31%	24/26	50%	3/6	75%	6/8	92.31%	24/26
test/unit/lib		100%	2/2	100%	0/0	100%	0/0	100%	2/2
test/utils		85%	17/20	100%	0/0	71.43%	5/7	88.24%	15/17

Figure 25



Backend end-to-end test coverage report

68.87% Statements `1655/2403` **49.7%** Branches `251/505` **54.6%** Functions `255/467` **69.48%** Lines `1621/2333`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲		Statements		Branches		Functions		Lines	
src		100%	7/7	66.67%	4/6	100%	0/0	100%	7/7
src/api/controllers		67.31%	105/156	78.57%	44/56	50%	17/34	67.31%	105/156
src/api/errors		75%	6/8	100%	0/0	0%	0/2	75%	6/8
src/api/middlewares		91.38%	53/58	27.27%	3/11	87.5%	7/8	91.38%	53/58
src/api/models		98.68%	150/152	80%	24/30	93.94%	31/33	98.57%	138/140
src/api/repositories		100%	68/68	50%	4/8	100%	2/2	100%	68/68
src/api/services		65.49%	186/284	72.5%	58/80	52.56%	41/78	66.3%	183/276
src/api/subscribers		100%	14/14	80%	4/5	100%	1/1	100%	14/14
src/auth		85%	51/60	58.82%	10/17	100%	11/11	85%	51/60
src/database/migrations		75.47%	80/106	100%	0/0	54.55%	24/44	75.47%	80/106
src/database/seeds		100%	95/95	100%	0/0	100%	8/8	100%	95/95
src/decorators		100%	20/20	100%	0/0	100%	6/6	100%	18/18
src/lib		63.64%	28/44	33.33%	2/6	68.75%	11/16	63.64%	28/44
src/lib/algorithms		66.67%	120/180	54.55%	12/22	50%	8/16	67.43%	118/175
src/lib/algorithms/gaComponents		65.61%	227/346	60.24%	50/83	72.73%	16/22	65.61%	227/346
src/lib/clustering		94.92%	56/59	70%	7/10	100%	4/4	94.92%	56/59
src/lib/database		58.87%	83/141	25%	4/16	70%	28/40	57.89%	77/133
src/lib/env		88.24%	15/17	50%	2/4	100%	4/4	88.24%	15/17
src/lib/graphql		29.34%	71/242	1.09%	1/92	3.17%	2/63	31.28%	71/227
src/lib/greedy		46.67%	42/90	50%	9/18	33.33%	3/9	47.19%	42/89
src/lib/logger		88.46%	23/26	66.67%	4/6	62.5%	5/8	88.46%	23/26
src/lib/seed		38.18%	42/110	0%	0/18	14.71%	5/34	38.78%	38/98
src/lib/shuffle		100%	8/8	100%	0/0	100%	1/1	100%	8/8
src/loaders		93.62%	44/47	50%	4/8	100%	8/8	93.62%	44/47
test/e2e/utils		97.67%	42/43	55.56%	5/9	87.5%	7/8	100%	39/39
test/unit/lib		100%	2/2	100%	0/0	100%	0/0	100%	2/2
test/utils		85%	17/20	100%	0/0	71.43%	5/7	88.24%	15/17

Figure 26

E Info Sheet

Title of the project: Dynamic Distribution through the city of Amsterdam.
Name of the client organisation: MakeTek
Date of the final presentation: 4-7-2018

Description:
The goal of this project is to make an automated system that solves bike delivery scheduling problems with limited bike carrying capacity, time windows, dynamic delivery additions, movable pickup points and delivery time estimates.
During the research phase we looked at similar problems in literature and found that a genetic algorithm was a very popular option. We also refined the client's requirements and decided what frameworks/libraries/boilerplates to use.
The project used an Agile workflow that included meetings with the client and TU supervisors every week. A full-stack development process was used to allow all members to experience the different parts of the development. The initial planning turned out to be too optimistic in terms of time costs of certain features, causing us to drop some features after discussion with the client.
The final product contains a backend that runs the genetic algorithm and can be reached through a RESTful API. The product also contains a visualiser that provides a user interface for the API. The solutions returned by the backend can be shown on a map on the visualiser. The product has been tested with unit, integration and end-to-end test suites that were consistently run on a continuous integration server.
The final product fulfils the requirements of the client and will be used.

Members of the project team:
Name: Ilja Bakx
Interests: Model Driven Rapid Prototyping and education
Contributions and roles: Developer, database interactions
Name: Jelle Eysbach
Interests: Algorithm Design, Agile workflow
Contributions and roles: Developer, visualisation
Name: Chris Mostert
Interests: Computational intelligence, data analytics
Contributions and roles: Developer, quality assurance
Name: Casper Schröder
Interests: Software architecture, machine learning
Contributions and roles: Developer, algorithm design
Client:
Stijn van Schooten, CTO at MakeTek
Supervisors:
Matthijs Spaan, Associate Professor at the Algorithmics Group of the Faculty of Engineering, Mathematics and Computer Science.
Johan Los, PhD at the Department of Maritime and Transport Technology.
Contact person:
info@maketek.nl
The final report for this project can be found at:
http://repository.tudelft.nl

F Original problem description

Dynamic Distribution through the city of Amsterdam

Network flow optimization of dynamic routes with an variable set of deliveries and deliverers

E-bike navigation Algorithm.
The current congestion problems, as a result of the overpressure in the traditional car delivery system within Amsterdam, is one of the biggest problem the city faces at this time. To resolve this problem, we At Maketek look for more sustainable options. Traversing through the city by bike offers new opportunities and risks, and optimizing these routes for bike lanes with the goal to reduce the traffic intensity within the center of Amsterdam.

Your project will be to create a real-time delivery scheduling system that implements the following features:

- Support for multiple bikes.
- Updated delivery time estimates.
- The navigation algorithm will use open source traffic data to take the least congested route.
- The delivery bikes will always be filled to capacity, and can dynamically add pickups and drops
- Creating Cliques of deliveries to be picked up by one bike.
- Scheduling Refill moments at Edge-of-city hubs.

Our end goal here is to share this data through an open platform, where Amsterdam based companies can work together to combine deliveries and reduce their time on the road, and consequently reduce their impact on traffic density and pollution and increase overall road safety within the city.

Project description:

Research proposed features as a feasibility report and create an algorithm for the features in the report. Implement the conclusion onto the existing package delivery system of Maketek. This is a challenging bachelor proposal where good planning is key to success. Affinity with the courses Software Engineering Methods &s Complexity Theory is recommended

Resources

Next to your TU Delft supervisor, you will be able to have guidance of an experienced systems architect, as well as a engineer from Fox-IT with 5+ years of experience. There will be weekly team meetings with the project supervisor where support to the team will be provided as best as possible. A more than sufficient budget is made available and can be spent in consultation with the project proposer. Office space is provided in the office of Maketek 10 minutes from campus, so the team can experience working in a start-up environment.

Project Proposer: Ilja Bakx (ilja@maketek.nl)

Project Supervisor: Stijn van Schooten (stijn@maketek.nl)