**The following should be included in your acknowledgements section:**

Map data copyrighted OpenStreetMap contributors and available from [https://www.openstreetmap.org](https://www.openstreetmap.org/)

**Reference:**

OpenStreetMap contributors. (2015) Planet dump [Data file from $date of database dump$]. Retrieved from [https://planet.openstreetmap.org](https://planet.openstreetmap.org/).

**OSM** OpenStreetMap

**GA** Genetic Algorithm

**GVR** Genetic Vehicle Representation

**OPEN DATA CHAPTER:**

Falar sobre à importancia de open data nas cidades a partir do governo e empresas. Falar sobre medidas colaborativas como o OpenStreetMap.

Development

This project aims to develop a genetic algorithm to deal with the problem of planning efficient routes to garbage trucks in a city. This problem, as stated before, is closely related with the CARP problem. But in real case scenarios, and in the Campolide’s case, those trucks don’t have all the same capacity CARP-MF. To deal with this difference some tweaks in the algorithms found in the literature must be made. This section gives an overview of the entire development of the project, problems faced, the solution applied to solve these problems and why these solutions were chosen.

The next subchapter will briefly contextualize the tools that were used to assist the problem. This is important to understand the structure of data that are gathered and explain the choices on how to deal with this data.

**A brief word on OpenStreetMap**

OpenStreetMap (OSM) is a collaborative mapping project where volunteers are free to create and edit geographic map information over the world to an open database. This database is also available for free under the Open Database License.

The street data from OSM, that are relevant to this project, are organized in nodes and ways. The node is the smallest point of data in the map, it represents a single point by defining its latitude and longitude. The nodes can also contain more information inside it, called tags, the tags are used to better qualify the node when it makes sense, for example if the node represent a pedestrian crossing or bus stops, this information will be refereed in tags. Some nodes have no tags, which is not a problem as they are used to represent a path.

<node id="21433116" visible="true" version="10" changeset="31019058" timestamp="2015-05-11T21:05:16Z" user="ddtuga" uid="1858517" lat="38.7244842" lon="-9.1772710"/>

Node example, user data omitted

Streets in the OSM are represented using one or more ways. Ways are ordered list of nodes, the way direction is defined by the order of the nodes in the way, having that the way starts at the first node and end at the last. Tags are also presented in ways to define things like the street name, the possible directions of the road, among other relevant information. Nodes shared between two or more ways define intersections between streets. Not all nodes are shared, there are nodes that are solely included in one way, for example, a node representing a pedestrian crossing may not also represent a street connection, among other cases.

<way id="233939235" visible="true" version="1" changeset="17392454" timestamp="2013-08-18T08:21:23Z" user="Bernhard W" uid="110838">

<nd ref="2422521543"/>

<nd ref="2422521485"/>

<nd ref="2422521542"/>

<tag k="highway" v="residential"/>

<tag k="lanes" v="1"/>

<tag k="name" v="Rua de Campolide"/>

<tag k="oneway" v="yes"/>

</way>

Way example, user data omitted

OSM is a powerful collaborative project that allow a variety of applications and studies on its data. There are much more about nodes, ways and how they relate to each other than discussed above, but that are not relevant to this project and therefore will not be addressed here.

In order to develop this project, data from Campolide streets are required, such as streets directions, length and connections, this information can be obtained using OSM. Data from OSM can be exported directly from their main website, but to this project a package called OSMnx was used. OSMnx is a Python package created by Boeing, G. (2017) that facilitates the download of administrative boundaries streets and perform a bunch of useful calculations using data from OSM.

Using OSMnx to download the data, a graph structure is retrieved with the nodes and ways representing the nodes and edges of a graph respectively. OSMnx perform data cleansing process automatically on downloads, the nodes that are not used in intersection and are presented in OSM ways are removed by an algorithm. This result in a simplified graph with just the relevant edges and nodes of a certain location.



Campolide representation with OSMnx

After the download of data from Campolide and Lisbon, both can be stored in the local machine using the library function *save\_graphml* to reduce the amount of time required to the upcoming runs of the algorithm, the graph is saved as a GraphML file into the disk.

**Distance Matrix**

The method to calculate the distances between edges in this project’s GA will follow the distance matrix proposed by Arakaki & Usberti (2018). The first step is to build a distance matrix between edges of the generated graph with Campolide’s ways. To build this matrix, each edge from the graph must have its shortest path value to the other edges, in the case of the edge to itself, the distance is defined as the length of the edge.

FIGURA DO GRAFO GERADO

In order to accomplish the creation of this Matrix, Lisbon graph data is needed above the Campolide’s one. In a city, from any point that a person is, he or she need to be able to get to any other point [REFERENCE: ???], if this is not possible, would have streets in the city that could not be reached by any means. In this case, Campolide is not an entire city, and is contained inside Lisbon, so it does not provide a path from one edge to all other edges. For that reason, a larger graph that contains the studied graph need to be used to be able to provide valid paths through its edges.

To calculate the distances, the method *shortest\_path\_length* from *network* package is used. Internally this method uses the Dijkstra’s algorithm to calculate the shortest path. The parameters required by the method are the graph, both nodes and the weight that must be taken in consideration, that in this case is the length of the way. The graph provided must be the wider one, the Lisbon graph, because of reasons already approached.

MORE TECHINICAL STUFF HERE NEEDED LIKE: The first step of HGA is to initialize the matrix of distances between required arcs. Let ER be the set of required edges and AR be the set of required arcs, where for each required edge {i, j} ∈ ER there are two corresponding required arcs (i, j), (j, i) ∈ AR. A matrix SP of dimensions |AR| × |AR| is computed such that each entry SP[e, f] is the shortest path cost from the ending node of arc e ∈ AR to the starting node of arc f ∈ AR. For sparse graphs (where |E| is much less than |V| 2) the SP can be computed within O(|V| 3) time and O(|V| 2) space by using the Floyd–Warshall algorithm (Cormen et al., 2001). The SP allows HGA to retrieve the distances between required arcs in O(1) time throughout the optimization process. [Hybrid genetic algorithm for the open capacitated arc routing problem]

FIGURA DE CAMPOLIDE COM AS EDGES FILTRADAS

Although using Lisbon graph as a parameter to the shortest path method, the edges used to iterate the process are from Campolide. Also, only edges that have tags with keys “highways” and values “residential” and “secondary” are taken in consideration as they are the ones where garbage trucks do regular collections. After this filter, 473 edges are left to be calculated, with 274 nodes. Every distance is measured in meters and the distances calculated are not distances between edges but nodes, and nodes distances do not make sense in the CARP problem that serves the edges. To solve this problem, the edges distances are calculated according to each of the shown cases below:

* Same edge: If the edges are the same, it means, the from and to edges are the same edge, the distance will be the total length of the edge. Basically, the distance between the first and last node;
* Different edges: For different edges, will be considered the last node from the first edge, and the first node of the last edge. Added to that are the previous calculated distances of the first and second edges alone. This sum gives the total distance between the two edges.

This process is repeated until every edge in the graph has a distance calculate to each other edge presented. It is a time-consuming task but done only once, with the results stored on the disk, since cities do not change the streets and its orientations often.

CODE

After this calculation, the distances between edges can be retrieved in O(1), as it just need to access the exact point in the matrix. These distances will be heavily used during the GA, so having the information with low complexity worth the time spent at first.

**GENETIC ALGORITHM**

Towards the implementation of a genetic algorithm, the representation of the problem, how to measure found solutions and the operations needed in order to generate the upcoming populations must be defined. In the literature review of genetics algorithms was seen the most common representations and operators of a genetic algorithm. This section will discuss how there these apply in this project.

A candidate solution in the population need to the comprehensive, it must carry organized data that allows it to be measured. This is important to the next subchapter that will discuss the fitness function created for this project. In the subsequent subchapters the initialization process and genetic operators will be addressed.

**Chromosome Representation**

This project will make use of a similar approach to the new GVR (Genetic Vehicle Representation) proposed by Pereira, Tavares, Machado & Costa (2002). Besides the fact that their approach deals with the VRP problem, this can receive small changes in order to fit this project’s problem.

The representation must contain the number of vehicles used, the served ways by each truck, as the order of the service. In this representation, the individuals are composed by a list of garbage trucks, each one containing an ordered subset of ways, each way listed in the set is a served edge by the corresponding truck. One solution must contain every edge that need to be served in the problem, independent of where which truck will serve it. Each chromosome is a valid solution to the problem.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Truck |  |  |  |  |  |
| 1 |  | ARC1 | ARC2 | ARC9 |  |
|  |  |  |  |  |  |
| 2 |  | ARC5 | ARC3 |  |  |
|  |  |  |  |  |  |
| 3 |  | ARC7 | ARC11 | ARC6 | ARC4 |
|  |  |  |  |  |  |
| 2 |  | ARC12 | ARC10 |  |  |
|  |  |  |  |  |  |
| 4 |  | ARC8 |  |  |  |

Chromosome example

In the example above, we have at least four trucks available to allocate in the routes. The number of ways that each truck is serving is related to its capacity. For simplification if every way demand is 1, the truck 1 can serve at least 3 ways, while the truck four at least 1. In the case of the capacity of the truck exceed, a new truck is chosen, and the remaining edges are allocated, the new truck must have enough capacity for at least the first edge that were not able to fit the previous truck, this process is repeated until every edge is allocated in a truck.

To represent this in a data structure in the program, a class representing the chromosome holds two arrays, one is the array with the path followed by the trucks, this array just keeps a sequence of ordered ways disregarding garbage trucks and capacities. The second array keep track of the trucks used in the solution, the first and last ways each one serve, their capacity and the load amount. A truck item in the second array always points to two subsequent indexes in the path array, this is how it keep track on the first and last way that it serves. It’s important to note that the index range referenced by the truck item is closed in the left side and open in the right [x1, x2).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ARC1 | ARC2 | ARC9 | ARC5 | ARC3 | ARC7 | ARC11 | ARC6 | ARC4 | ARC12 | ARC10 | ARC8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (1,5), 0, 3, 3 | (2,2), 3, 5, 2 | (3,4), 5, 9, 4 | (2,2), 9, 11, 2 | (5,1), 11, 12, 1 |

(Truck\_id, capacity), start\_index, end\_index, load

This combination of arrays provides the information where each truck serves one or more ways in the solution. This representation also allows the genetic operator to act simpler in the paths array, just needing the trucks array to be updated accordingly.

**Fitness function**

**Initialization**

**Operations**

**Elitism**

**Selection**

**Crossover**

**Mutations**