ERASMUS UNIVERSITEIT ROTTERDAM

Master Thesis
Operations Research and Quantitative Logistics

Erasmus School of Economics

Date Final Version:
April 27, 2022

Algorithms for the Edge Orientation Problem

Name Student: Anne de Vries
Student Number: 483897

Supervisor: dr. Wilco van den Heuvel
Second Assessor: dr. Twan Dollevoet

Abstract

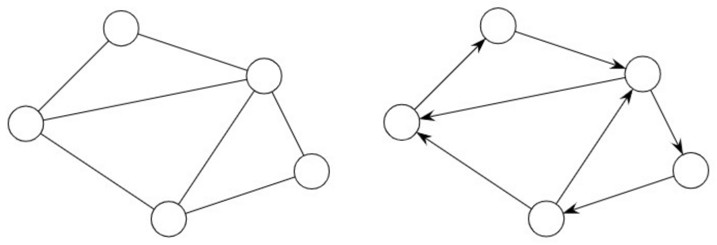
The Edge Orientation Problem (EOP) is rooted in the need for one-way walking directions during the Covid-19 pandemic. The EOP entails assigning an orientation to each edge of an undirected graph, such that each node is reachable by all other nodes and that the sum of all the shortest paths is minimised. To the best of our knowledge, this specific problem has not been studied in the literature before. We provide a proof NP-completeness of a problem related to the EOP. Moreover, we develop a number of solutions techniques for the EOP. We show that a Benders Decomposition can, when problem specific knowledge is applied, handle larger instances than the commercial solver CPLEX. We also develop two heuristics for the problem; a Genetic Algorithm, which is particularly fast and leaves small gaps with the optimal solution, and a hybrid between a Variable Neighbourhood Search and a Tabu Search. This last heuristic finds 28 of the 30 best known solutions in a short amount of time.

Contents

1 Introduction

Amidst the Covid-19 pandemic, a broad scale of measures were taken in order to reduce viral transmissions. One of these measures consisted of assigning one-way walking directions to paths in cities and in buildings. The question arose how these directions can be assigned efficiently. This research considers these problems from the point of view of graph theory by considering the Edge Orientation Problem (EOP).

In the EOP, a graph with undirected edges has to be orientated. An orientation is an assignment of a single direction to each edge and thus transforming the edges into arcs. In this EOP framework, there are many different variants of the problem. We consider the version of the EOP where each edge has to be orientated such that each node in the graph is reachable by all other nodes. Moreover, the objective is to minimise the total distance of the shortest paths. An example of an solution is presented in Figure 1. The problem we consider is, to the best of our knowledge, not studied in the literature. Although the literature on the EOP is sparse in general, some problems related to ours are studied. For example, the EOP for a set of Origin-Destination pair (OD-pair) and problems concerning the reachability in graphs are studied. In the literature there is a focus on proofs of complexity and theoretical properties.



(a) The undirected graph. (b) The optimal EOP solution for the graph.

Figure 1: An example of an optimal solution to the EOP.

This thesis adds to the literature by providing a new proof for the NP-completeness of an existing problem. Moreover, we introduce a new problem variant and introduce four solution techniques. We generate 30 instances of the problem for testing. The solution techniques include two versions of a Benders Decomposition. The first does not use any problem specific knowledge. This version performs poorly due to weak cuts and high memory usage. The second version does use problem specific knowledge, for example by partitioning the subproblem such that cuts can be generated and added per OD-pair and by including initial feasibility cuts. Although the commercial solver CPLEX performs better for smaller instances, this version of the Benders Decomposition uses less memory and therefore can handle much larger instances.

Besides these exact methods, two metaheuristics are applied to the problem. A Genetic Algorithm (GA) for the EOP is developed. This GA is fast, as it requires just two minutes to find solutions for all 30 instances. For the 15 instances with a known optimum, the GA found the optimal solution for 9, while for the other

6 instances the gap was on average 1.1%. For the 15 large instances, the upper bound found by the exact methods was on average improved by 6.4%. The second heuristic approach is a hybrid between a Variable Neighbourhood Search and a Tabu Search (VNS/TS). This approach is slower, requiring 9 minutes to find solutions for all 30 instances. Its solutions however, are of a high quality. It found the optimal solution for 13 of the 15 instances for which the optimum is known. For all the 15 largest instances, the best known solution was generated by the VNS/TS. It improved the upper bounds of the exact methods by 11.3% on average and the solutions from the GA by 5.3% on average.

The remainder of this thesis is as follows: In Section 2, the most important related literature is discussed. Then the EOP is thoroughly introduced in Section 3. Moreover, in this section, a mathematical model for the EOP is presented. Section 4 contains a proof for the NP-completeness of a problem related to the EOP. In Section 5, we introduce the four solution methods we have implemented. The performances of these methods are presented in Section 6. Lastly, we summarize and discuss our findings in Section 7.

2 Literature review

The literature on the Edge Orientation Problem (EOP) is sparse and spread out over many variations of the problem. In this section, we will provide a short overview of the most important related works.

A fundamental paper in this field is Robbins (1939). This paper is set in the context of a road network with two-way streets, but due to occasional high demand, streets can be assigned a one-way orientation. This orientation has to be such that one can travel from each point in the network to any other point. The paper proves that this can be done, in other words that the graph is orientable, if and only if the (undirected) graph is still connected after the removal of any arbitrary edge. This condition can be reformulated as that the graph has to be a two-edge-connected graph. This theorem is known as Robbins' theorem and is a necessary condition for the feasibility of the EOP considered in this paper. The theorem is extended to mixed graphs by Boesch & Tindell (1980).

Most related to our research is Ito, Miyamoto, Ono, Tamaki, & Uehara (2013). They consider two problems which they call Route-Enabling Graph Orientation Problems. In such a problem, a graph and a set of OD-pairs is given. In one problem, which they call MIN-MAX ORIENTATION, the objective is to orientate the graph such that the maximum shortest path distance among all OD-pairs is minimized. In the other problem, which they call MIN-SUM ORIENTATION, the objective is to minimise the sum of the shortest distance among the OD-pairs. This latter problem is similar to ours, however, we consider all possible pairs, instead of a number of OD-pairs. Ito et al. (2013) prove that the problems are NP-hard for planar graphs. Moreover, they provide algorithms for the problems in case the graphs are cacti.

Hassin & Megiddo (1989) study the ideal orientation for a given set of pairs. An ideal orientation is an orientation which does not disturb the shortest path for the pairs in the set. Hassin & Megiddo (1989) provide an polynomial time algorithm for the case of two pairs and prove that the problem is NP-complete for a general number of pairs. The paper also proves NP-completeness of some related problems. One of these problems is the decision problem of recognising whether there exists an orientation of a graph such that the shortest path for two pairs are not larger than a certain value.

The effect of orientation on the diameter of a graph is studied by Chvátal & Thomassen (1978). The diameter of a graph is the greatest shortest distance in the graph. They prove that a 2-edge-connected graph with diameter d can be orientated such that the diameter of the resulting graph is no larger than $2d^2 + d$.

Some other Edge Orientation Problems are only interesting for graphs for Robbins' theorem does not apply. Hakimi, Schmeichel, & Young (1997) regard the problem of finding the orientation that maximises the number of pairs that are reachable. The paper shows how the one-edge connected graph can be reduced to a weighted tree and gives a quadratic algorithm for the problem. Arkin & Hassin (2002) also consider connectivity requirements, but on mixed graphs instead of strictly undirected graphs.

Research in orientation, connectivity, and reachability has application in biology as Silverbush et al. (2011) and Gamzu et al. (2010) apply it to interactions between proteins and DNA in a cell. These papers also provide approximations algorithms and formulations for the problems they consider.

3 Problem description

In this section, the Edge Orientation Problem is explained more thoroughly. Then, we present a mathematical model for the problem.

The Edge Orientation Problem considers a graph with undirected edges. To each of these edges, an orientation has to be assigned. This means that each undirected edge is replaced by a directed arc. This has to be done in such a way that from each vertex, all other vertices are reachable. The objective is to minimise the sum of the distances of the shortest paths for each OD-pair.

More formally, the simple graph $G = (V, E)$, with V the set of vertices and E the set of undirected edges, is considered. For each edge (i, j) an orientation is decided using two binary variables, $u_{i,j}$ and $u_{j,i}$. Only one of these variables can be set to one, meaning that the edge can only be traversed in the direction dictated by the subscript. Each edge has a non-negative length $d_{i,j}$ and we assume that the directed arc associated with the edge has the same distance, irrespective of its orientation. Orientating all edges in E yields a set of directed arcs, denoted A. With set A, the directed graph $G' = (V, A)$ can be created. In graph G', there must be a feasible path between each ordered pair of vertices. This path is found using the binary variable $x_{i,j}^{s,t}$, which equals to one if the arc (i, j) is part of the shortest path from s to t. Clearly, an arc can only be traversed in the direction of its orientation. Table 1 provides an overview of the various symbols used.

Table 1: An overview of the sets, parameters, variables, and graphs used in the mathemetical formulation of the Edge Orientation Problem.

Symbol	Type	Description
G	Graph	The undirected graph
G'	Graph	The directed graph
V	Set	The set of vertices
E	Set	The set of (undirected) edges
A	Set	The set of (directed) arcs
$d_{i,j}$	Parameter	The distance of the edge of arc between node i and j
i	index	A node
j	index	A node
s	index	The destination node of an OD-pair
t	index	The origing node of an OD-pair
$u_{i,j}$	Decision Variable	Indicates if the edge between i and j is directed from i to j
$x_{i,j}^{s,t}$	Decision Variable	Indicates if the arc from i to j is used for the path from s to$t$

The objective of the problem is to find the orientations and paths such that the total distance is minimised. Note that this objective is congruent with minimising the average distance between pairs. A mathematical model, specifically a Mixed Integer Linear Programming formulation, for this problem is as follows:

$$\min \quad \sum_{i,\,j\,\in\,V} d_{i,j}x_{i,j}^{s,t} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\,\in\,V} x_{i,j}^{s,t} - \sum_{j\,\in\,V} x_{j,i}^{s,t} = 0 \qquad \forall i,s,t \in V,\ s \neq t,\ s \neq i,\ t \neq i \tag{2}$$

$$\sum_{j\,\in\,V} x_{s,j}^{s,t} = 1 \qquad \forall s,t \in V,\ s \neq t \tag{3}$$

$$\sum_{j\,\in\,V} x_{j,t}^{s,t} = 1 \qquad \forall s,t \in V,\ s \neq t \tag{4}$$

$$x_{i,j}^{s,t} \leq u_{i,j} \qquad \forall i,j,s,t \in V,\ s \neq t \tag{5}$$

$$u_{i,j} + u_{j,i} = 1 \qquad \forall i,j \in V \tag{6}$$

$$x_{i,j}^{s,t} \in \{0,1\} \qquad \forall i,j,s,t \in V,\ s \neq t \tag{7}$$

$$u_{i,j} \in \{0,1\} \qquad \forall i,j \in V \tag{8}$$

In this model, the objective of minimising the sum of distances of the shortest paths is defined by Equation (1). Constraint sets (2) through (4) are the flow conservation constraints for the shortest paths. Constraint set (5) ensures that flow only occurs in direction of the orientation and Constraint set (6) makes sure that only one orientation is chosen for each edge. Constraints sets (7) and (8) indicate the ranges of the decision variables.

This formulation can be relaxed as the variables $x_{i,j}^{s,t}$ are flow variables. Given that the $u_{i,j}$ variables are binary and that there are no capacities, there is only one shortest distance for each OD-pair. In case an orientation has two different, feasible paths with the same (shortest) distance for an OD-pair, either one can be chosen without altering the orientation. Therefore, the Constraint set (7) can be replaced with:

$$0 \leq x_{i,j}^{s,t} \leq 1 \qquad \forall i, j, s, t \in V,\ s \neq t \tag{9}$$

The mathematical model is very flexible and can accommodate many other variants of the Edge Orientation Problem with minor adjustments. For example, the objective function can be changed such that the worst-case (i.e. the longest shortest path) is minimised or the constraints can be altered such that a certain number of edges is allowed to remain without orientation. Another option is, instead of considering all pairs, introducing a set of relevant pairs which need to be reachable and have their shortest path minimised.

4 NP-completeness

In this section, a proof for the NP-completeness of a generalization of the EOP is given. Namely, we proof that the following decision problem is NP-complete:

Problem 1. Given an undirected graph G, a set of node pairs P and an value K, recognise whether there exists an orientation G' such that there exists a path between all node pairs in P, and that the sum of the distances of those paths is shorter than or equal to K.

This problem is also a generalisation of the optimisation version of the MIN-SUM ORIENTATION problem considered in Ito et al. (2013). They provide a proof for the NP-hardness of their problem. Therefore, a proof by reduction would suffice to prove the NP-completeness of Problem 1. However, we provide a simpler proof then that in Ito et al. (2013).

Proposition 1. *Problem 1 is NP-complete.*

Proof. A solution can be verified in polynomial time, for example with the Floyd-Warshall algorithm, which is introduced by Floyd (1962), and thus the problem belongs to the class NP. Next, we prove polynomial reduction from the 3 Satisfiability Problem (3SAT), which is proven to be NP-complete by Karp (1972). An instance of this problem consists of a set of boolean variables $U = \{u_1, ..., u_n\}$ and a set of clauses $C = \{c_1, ..., c_m\}$. A clause consists of at most three literals. A literal is either a variable or a negation of one. A clause is satisfied if at least one of its literals is true. If all clauses are satisfied, the instance is feasible.

An instance of the 3SAT Problem can be transformed in an instance of Problem 1 as follows; For each variable u_i, create six nodes, denoted $(i1+), (i2+), (i1-), (i2-), (iO)$ and (iD) and six edges, namely $(i1+) - (i2+)$, $(i1-) - (i2-)$, $(iO) - (i2+)$, $(iO) - (i2-)$, $(i1+) - (iD)$ and $(i1-) - (iD)$. Also add (iO) and (iD) to the set P, such that there has to exists a directed path from (iO) to (iD). For each clause c, create two nodes, denoted (CiO) and (CiD). If positive literal i is a member of c, add the edges $(CiO) - (i1+)$ and $(i2+) - (CiD)$. Similarly, if a negative literal i is a member of c, add the edges $(CiO) - (i1-)$ and $(i2-) - (CiD)$. Also add (CiO) and (CiD) to the set P. Lastly, give each edge an edge length of one and set $K = 3|P|$. An example of a transformed instance is given in Figure 2.

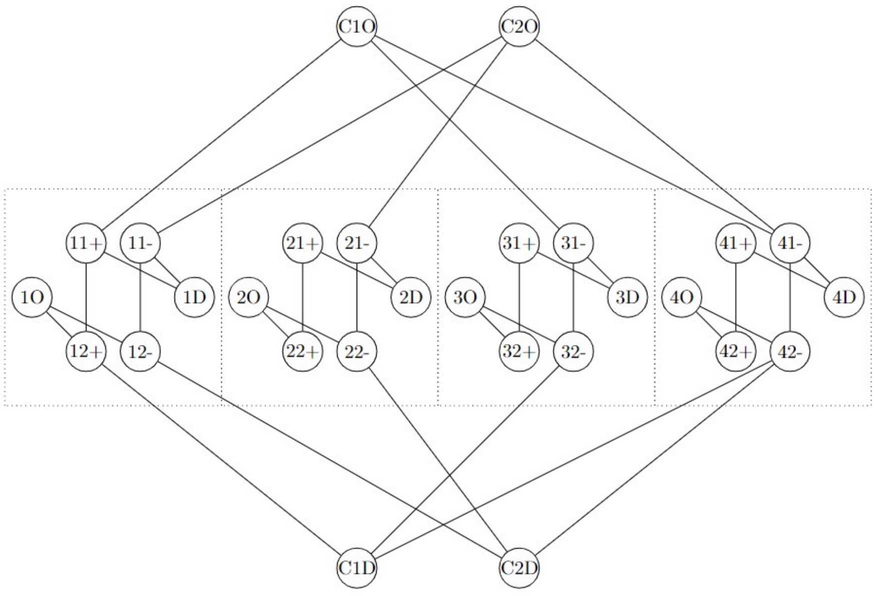


Figure 2: An example of a 3SAT instance transformed to an instance of Problem 1. The 3SAT instance consists of variales $U = \{u_1, u_2, u_3, u_4\}$ and clauses $C = \{\{u_1, \overline{u}_3, \overline{u}_4\}, \{\overline{u}_1, \overline{u}_2, \overline{u}_4\}\}$.

We now need to prove that an instance of 3SAT is a 'yes' instance if and only if its corresponding instance of Problem 1 also is a 'yes' instance. We do this by proving that a solution of a 'yes' instance of 3SAT can be transformed into a feasible solution of Problem 1 and vice versa.

First, assume we have a 'yes' instance to the 3SAT problem. We can transform this to a solution to the corresponding instance of Problem 1 as follows; For all variables i, orientate the necessary edges such that the path from (iO) and (iD) goes via the negative nodes if the variable is positive and vice versa. These paths all have length 3. Next, for all clauses c orientate the necessary edges such that there is a path from (CiO) and (CiD) via the edge associated with the satisfied literals of the clause. These paths also have length 3 and will not interfere with the previous orientation because the 3SAT instance is satisfied. Next, orientate the edges $(i1+) - (i2+)$, $(i1-) - (i2-)$ to the arcs $(i1+)- > (i2+)$, $(i1-)- > (i2-)$ if no other orientation was decided. The remaining edges can be orientated randomly and will not cause paths shorter than length 3 between any pair in P. Therefore, we have that the combined distance of the paths is equal to $3|P| = K$.

Next, assume that we have a feasible solution for an instance of Problem 1. A solution to the corresponding 3SAT instance can be obtained by setting all variables i to TRUE if there is an arc $(i1+)- > (i2+)$ and to FALSE otherwise. As there exists a directed path of length 3 between (CiO) and (CiD), every clause will have at least one member satisfied. Therefore the 3SAT instance is satisfied. □

5 Methodology

This section introduces multiple algorithms for solving the EOP. In Section 5.1, a construction heuristic, which provides quick but possibly infeasible solutions, is introduced. Next, Section 5.2 introduces the Benders Decomposition, which can guarantee optimality. Section 5.3 and Section 5.4 provide the Genetic Algorithm (GA) and a hybrid between a Variable Neighbourhood Search and a Tabu Search (VNS/TS), respectively. These two solution techniques are heuristics, which are generally used to get solutions fast, without a guarantee of optimality.

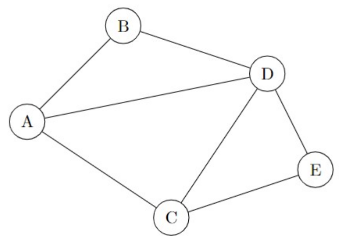
5.1 Construction Heuristic

We first introduce a construction heuristic. It quickly provides a solution from scratch. These solutions may not be feasible. However, they provide a good starting point for further methods. Therefore, all other methods introduced in Section 5 use this construction heuristic. The rationale of the construction heuristic is to create cycles in the graph, as cycles allow nodes to reach and to be reached by all nodes in the cycle and therefore are a good spine of a solution. Algorithm 1 provides an overview of the construction heuristic. The details will be discussed below.

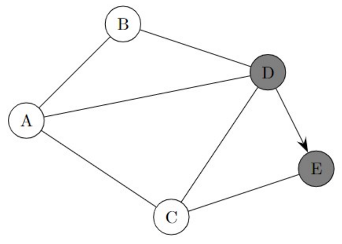
Algorithm 1: Overview of the Construction Heuristic

```
1  Input: NumberOfCandidates = an integer number of candidate edges to consider
2  active ← empty set
3  e ← a random edge
4  give a random orientation to e
5  add the nodes associated with e to active
6  while not all edges are orientated do
7      candidates ← the NumberOfCandidates shortest, undirected edges between a
         node in the set active and a node outside it
8      e ← a random edge from candidates
9      n ← the node connected to e not in candidates
10     orientate e towards n
11     for all undirected edges i from n to nodes in active do
12         orientate i away from n
13     end
14     add n to active
15 end
```

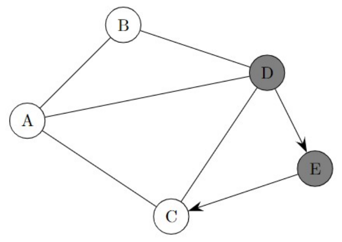
The construction heuristic takes one parameter, namely *NumberOfCandidates.* This parameter denotes the number of candidates considered in the restricted candidate list. The algorithm begins by initializing a set of nodes named *active*. A random edge is chosen and is given a random orientation. The two nodes connected to this edge are added to *active*. Then, the following procedure is repeated until there are no edges left without an orientation; a restricted candidate list of the shortest edges is made. For this list, only edges between one node in and one node not in the *active* set are considered. A total of *NumberOfCandidates* candidates are considered. Of these candidates, a single edge e is randomly chosen. Let node n be the node which is attached to e but not in *active*. Edge e is orientated towards n. Furthermore, all edges without orientation between n and nodes in *active* are orientated away from n. Then, n is added to *active*. An example of how the construction heuristic makes a solution is provided in Figure 3.



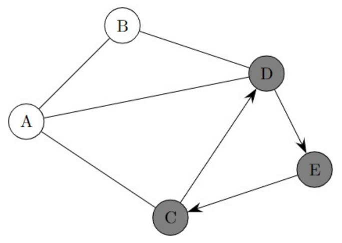
(a) The undirected graph which will get a solution generated by the construction heuristic.



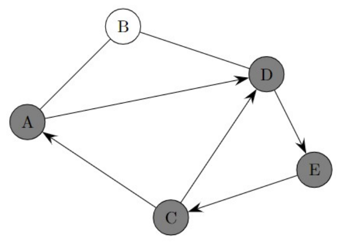
(b) A random edge is orientated and the associated nodes are added to *active*. [2 - 5]



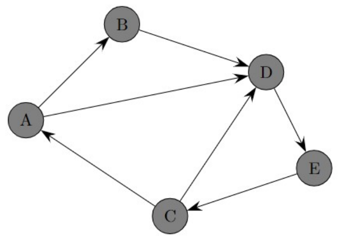
(c) The first part of the first iteration; An edge attached to exactly one node in *active* is chosen and orientated away from the node in that set. [7 - 10]



(d) The second part of the first iteration; The newly connected node is added to the set *active* and edges between it and other nodes in that set are orientated away from the newly added node. [11 - 14]



(e) The second iteration; Two edges are orientated and one node is added to *active*. [6 - 15]



(f) The third and final iteration; As the remaining two edges are orientated, a complete solution is acquired. [6 - 15]

Figure 3: An example of how the construction algorithm works. The edge lengths are the Euclidean distance. In this example, $NumberOfCandidates$ is set to 1. In other words, an iteration orientates the shortest edge without orientation which is attached to *active*. Nodes in the set *active* are coloured grey. The square brackets in the captions of the subfigures indicate which lines of Algorithm 1 are performed.

5.2 Benders Decomposition

We implement a Benders Decomposition to solve the EOP. Benders Decomposition is a solution strategy introduced by Benders (1962). The strategy works for problems with a complicated part and an easier part. Instead of jointly solving both parts, a Benders Decomposition solves both parts iteratively, while transferring information about the optimal solution between the parts. The hard part, called the Benders Master Problem (BMP), is solved first. This is faster than solving the complete problem as it is smaller and has less constraints. The solution found for the complicated part is then fixed and used to solve the easier part. As this part is easy, a solution can be found quickly for the subproblem. This solution to the easy part contains information about the feasibility and the optimality of the found solution to the joint optimisation problem. This information can be added as a constraint to the BMP, called a cut. This procedure is performed until the subproblem does not find any cut. If no cut can be found, the optimal solution to the joint problem is guaranteed to be found.

In general, the variables related to the hard part are denoted y and those related to the easy part are denoted x. The structure needed to solve a problem with Benders Decomposition is as follows:

$$\begin{aligned} \min \quad & c^T x + f(y) && (10) \\ \text{s.t.} \quad & Ax + g(y) \geq b && (11) \\ & x \geq 0 && (12) \\ & y \in S && (13) \end{aligned}$$

where A is a matrix, c and b is a vector and $f(y)$ and $g(y)$ are functions, not necessarily linear. S is a set containing all possible values of y. A problem with such a structure can be reformulated. For the details of this reformulation we refer to Rahmaniani, Crainic, Gendreau, & Rei (2017). The reformulated problem, which is the BMP, is as follows:

$$\begin{aligned} \min \quad & z && && (14) \\ \text{s.t.} \quad & z \geq f(y) + (b - g(y))^T \hat{u}_i && i \in I_p && (15) \\ & 0 \geq (b - g(y))^T \hat{v}_i && i \in I_c && (16) \\ & y \in S && && (17) \end{aligned}$$

where I_p is a set of optimally cuts and I_c is a set of feasibility cuts. The vectors $\hat{u}_i$ and $\hat{v}_i$ are extreme points and extreme directions of the polyhedron defined by $A^T u \leq c$ and $u \geq 0$, respectively.

In general, this polyhedron has many of these extreme points and extreme directions. It is therefore computationally cumbersome to find all of them and then solve the reformulated problem. Another approach is to solve the BMP while I_c and I_p are empty sets and then solve a subproblem to find the most violated constraint. This constraint is then added as a cut to the BMP. This process is repeated until no violated constraint can be found, in which case the optimal solution is found. As the number of extreme directions and extreme points is finite, the procedure is guaranteed to end.

The most violated constraint can be found as using the solution (z^*, y^*) of the BMP. This solution is used in the subsequent subproblem. Solving this subproblem is also referred to as row generation. The subproblem is as follows:

$$\begin{aligned} \max \quad & (b - g(y^*))^T u && (18) \\ \text{s.t.} \quad & A^T u \leq c && (19) \end{aligned}$$

$$u \geq 0 \tag{20}$$

where u is a vector of decision variables, which represent the dual variables of x. Solving this problem either yields a optimal solution or an extreme direction. These can be used to obtain a cut as follows; If the problem is unbounded, a extreme direction $\hat{v}_i$ is found and added as a feasibility cut in I_c. If the problem is bounded we found an extreme point $\hat{u}_i$. If we have $f(y^*) + (b - g(y^*))^T \hat{u}_i > z^*$, we can add $\hat{u}_i$ to the set of optimally cuts I_p. As aforementioned, in case a cut was found the procedure is repeated, starting with solving the BMP. Otherwise, the optimal solution to the original problem is found. This solution has objective value z^* and the optimal values for y are y^*. The optimal values for x can be found by solving the original problem with $y = y^*$ or by using the Complementary Slackness Theorem.

To apply Benders Decomposition to the EOP requires a slight alteration to the formulation of the problem in Section 3 such that it has the structure of the problem defined by Equations (9)-(12). The reformulated problem is as follows:

$$
\begin{aligned}
\min \quad & \sum_{s,t \in V,\ s \neq t} \sum_{i,j \in V} d_{i,j} x_{i,j}^{s,t} && && (21)\\
\text{s.t.} \quad & \sum_{j \in V} x_{i,j}^{s,t} - \sum_{j \in V} x_{j,i}^{s,t} \geq 0 && \forall i,s,t \in V,\ s \neq t,\ s \neq i,\ t \neq i && (22)\\
& -\sum_{j \in V} x_{i,j}^{s,t} + \sum_{j \in V} x_{j,i}^{s,t} \geq 0 && \forall i,s,t \in V,\ s \neq t,\ s \neq i,\ t \neq i && (23)\\
& \sum_{j \in V} x_{s,j}^{s,t} \geq 1 && \forall s,t \in V,\ s \neq t && (24)\\
& \sum_{j \in V} -x_{s,j}^{s,t} \geq -1 && \forall s,t \in V,\ s \neq t && (25)\\
& \sum_{j \in V} x_{j,t}^{s,t} \geq 1 && \forall s,t \in V,\ s \neq t && (26)\\
& \sum_{j \in V} -x_{j,t}^{s,t} \geq -1 && \forall s,t \in V,\ s \neq t && (27)\\
& -x_{i,j}^{s,t} + u_{i,j} \geq 0 && \forall i,j,s,t \in V,\ s \neq t && (28)\\
& 0 \leq x_{i,j}^{s,t} \leq 1 && \forall i,j,s,t \in V,\ s \neq t && (29)\\
& u_{i,j} + u_{j,i} = 1 && \forall i,j \in V && (30)\\
& u_{i,j} \in \{0,1\} && \forall i,j \in V && (31)
\end{aligned}
$$

This formulation differs from the previous one in that Constraint sets (22) though (27) are split such that they have the greater-than-equal-to structure found in the reformulated problem defined Equations (14)-(17). The $u_{i,j}$ variables are the difficult part (the y variables). The feasible set S is described by constraint sets (30) and (31). Furthermore, we have $f(y) = 0$ and $g(y)$ is linear and therefore can be represented as vector. The variables $x_{i,j}^{s,t}$ make up the easier part (the variables x).

5.2.1 Problem Specific Improvements

Knowledge about the EOP allows for tailoring the Benders Decomposition such that the performance in speed and memory usage is improved. Below we will name observations about the EOP and how this knowledge can be used to improve the performance. We will start with what can be done before the first iterations and follow by how the row generation can be improved for the EOP.

To solve the EOP with a Benders Decomposition, various matrices and vectors which define the problem and are used throughout the Benders Decomposition are needed. These get large and occupy memory. However, they are sparse and their structure is rather simple. Therefore, their logic can be implemented as a function instead of storing the entire arrays to reduce memory usage. An example of this is the vector b, which solely contains zeros (from Constraint sets (22), (23), and (28)), positive ones (from Constraint sets (24) and (26)), and negative ones (from Constraint sets (25) and (27)). Based on the size of the instance, the correct value b can be calculated for each index.

Knowledge about the problem also allows us to add initial constraints to the Master Problem. First, we observe that a solution to the EOP can only be feasible if a node can reach and is reachable by all nodes. Therefore each node must have at least one edge orientated towards the node and one edge orientated away from that node. Therefore, we can add for each node two constraints to the BMP to enforce this. The constraints are:

$$\sum_{j \in V} u_{i,j} \leq q_i - 1 \qquad \forall i \in V \tag{32}$$

$$\sum_{j \in V} u_{j,i} \leq q_i - 1 \qquad \forall i \in V \tag{33}$$

where q_i denotes the number of edges connected to node i. Adding these constraints makes the set S tighter and reduces the number of iterations which result in an added feasibility cut.

Another observation is that flipping all orientations of a solution to the EOP results, because all OD-pairs are considered, in a solution with an identical objective value. By fixing the orientation of a single edge, the solution space is smaller while still containing an optimal solution. This leads to fewer iterations. Lastly, a solution generated by the Construction Heuristic presented in Section 5.1 can be used as starting point.

The row generation can also be sped up significantly. To see this, it is important to realise what the subproblem is. It is the dual problem of the problem of finding the shortest paths for all OD-pairs, or the all-to-all shortest path problem, given the orientation found in the Master Problem. This observation leads to partitioning the subproblem into the dual of the shortest path problem for each pair, which reduces memory usage.

Next, the same observation about the subproblem allows us to split the variable for the objective function z into a variable for each OD-pair $z^{s,t}$. This variable represents the length of the path between s and t. The objective value then becomes the sum of all these variables; $\sum_{s,\, t\, \in\, V,\ s\, \neq\, t} z^{s,t}$. The subproblem becomes:

$$\begin{aligned} \max \quad & (b_{s,t} - g_{s,t}(y^*))^T u_{s,t} && (34) \\ \text{s.t.} \quad & A_{s,t}^T u_{s,t} \leq c_{s,t} && (35) \\ & u_{s,t} \geq 0 && (36) \end{aligned}$$

Because of this change, cuts can be added for each pair separately. This makes them stronger and results in needing to perform less iterations.

Finally, the Floyd-Warshall algorithm can solve the (primal) all-to-all shortest path problem. Therefore, it can decide which OD-pairs are unbounded or have a feasible solution in the dual. If an OD-pair is unbounded in the dual, the partial subproblem only needs to be solved for this pair. If an OD-pair has a feasible solution, the Floyd-Warshall algorithm also provides the length of that path. If that length is equal to the value of the $z^{s,t}$ variable for that pair, that partial subproblem does not need to be solved, as no optimality cut can be found. Implementing this into a Benders Decomposition reduces the number of partial subproblems that need to be solved and therefore speeds up the row generation.

5.2.2 Implementation

We implement two versions of the Benders Decomposition. The first, which we will refer to as general Benders Decomposition, uses few of the improvements based on the problem specific knowledge. Namely, it does use the memory improvements, such that we are able to compare more results. The second version implements all improvements based on the problem specific knowledge. We refer to this version as the specialised Benders Decomposition.

For both versions, we use the state of the art solver CPLEX. To increase the performance, we make use of the “Lazy Constraint” feature this solver offers. Lazy constraints are constraints which affect the feasible region but are, according to the user, unlikely to be violated. We can add the optimality and feasibility cuts generated in the subproblem as lazy constraints. The advantage of this is that, contrary to solving the BMP and the subproblem as standalone models, adding cuts as lazy constraints does not require the branch-and-bound tree of the BMP to be restarted, speeding up the execution. More information on lazy constraints can be found in CPLEX, IBM ILOG (2017).

5.3 Genetic Algorithm

A GA is a metaheuristic inspired by natural selection in biology. Its basic principle is to generate a group of solutions and recombining the best solutions of this group to generate a new group.

To apply a GA, solutions should be able to be represented as a chromosome. A group of chromosomes is called a population. Each chromosome can be attributed a fitness score, which is the objective value of the solution, penalized for infeasibilities. Chromosomes are selected for recombination, a process in which new chromosomes are generated, based on their fitness. Those with a better fitness value, have a higher probability of getting selected. This provides pressure towards better solutions. After the recombination, there is a probability of mutation, which ensures diversification. The new chromosomes are grouped into a new population such that the process can be repeated.

A GA can be applied to the EOP, as EOP solutions are easy to represent as a chromosome. More specifically, an EOP solution can be represented as a binary string with a gene for each edge to indicate its orientation. Algorithm 2 provides an overview of the genetic algorithm we implement. The details are discussed below.

Algorithm 2: Overview of the Genetic Algorithm

```
1  Input: $popSize \leftarrow$ number of chromosomes in a population
2  Input: $tournamentSize \leftarrow$ number of chromosomes partaking in a tournament
3  Input: $mutChance \leftarrow$ the probability of a chromosome being exposed to mutation
4  Input: $mutChanceConditional \leftarrow$ the probability of a gene mutating given that the
     chromosome is exposed to mutation
5  Input: $\eta_{GA} \leftarrow$ maximum number of iterations without improvement
6  Input: $NumberOfCandidates \leftarrow$ Size of the restricted candidate list in the
     construction heuristic,
7  Input: $\alpha_0 \leftarrow$ starting value of the penalty function
8  $\alpha \leftarrow \alpha_0$
9  for $1$ to $popSize$ do
10     $chromosome \leftarrow ConstrutionHeuristic(NumberOfCandidates)$
11     $individual \leftarrow FitnessValue(chromosome, \alpha)$
12     $population \leftarrow population \cup individual$
13 end
14 $generation \leftarrow 0$
15 $bestGeneration \leftarrow 0$
16 $penaltyCounter \leftarrow 0$
17 while $generation - bestGeneration < \eta_{GA}$ do
18     $generation \leftarrow generation + 1$
19     if there is a new best chromosome in population then
20         $bestGeneration \leftarrow generation$
21         store the best solution in $bestSolution$
22     end
23     $UpdatePenalty(\alpha, penaltyCounter, population)$
24     $newPopulation \leftarrow$ a new, empty population
25     while $|newPopulation| < popSize$ do
26         $P_1, P_2 \leftarrow TournamentSelection(tournamentSize)$
27         $O_1, O_2 \leftarrow Recombine(P_1, P_2)$
28         $Mutate(O_1, mutChance, mutChanceConditional)$
29         $Mutate(O_2, mutChance, mutChanceConditional)$
30         $C_1 \leftarrow FitnessValue(O_1, \alpha)$
31         $C_2 \leftarrow FitnessValue(O_2, \alpha)$
32         $newPopulation \leftarrow newPopulation \cup C_1 \cup C_2$
33     end
34     $population \leftarrow newPopulation$
35 end
```

The fitness of an individual is determined by the sum of the distances of the shortest paths between all pairs plus a penalty if no such path exists. The distances can be calculated with the the Floyd–Warshall algorithm. For the penalty value, we use an adaptive penalty function similar to the one discussed in Smith, Coit, Baeck, Fogel, & Michalewicz (1997). A penalty α is added to the fitness function for each infeasible pair. The value of α is initialized at α_0. However, if for η_α consecutive iterations the fittest solution of a population has infeasibility penalties, α is multiplied by δ_α. Conversely, if the best solution of η_α consecutive iterations does not encounter a penalty, α is divided by δ_α. As the EOP is a minimisation problem, a lower fitness value indicates a fitter individual. Algorithm 3 provides an overview of how the penalty values are updated.

Algorithm 3: Overview of the updating of penalty values in the GA

```
1  Input: $\alpha \leftarrow$ the current penalty factor
2  Input: $\eta_\alpha \leftarrow$ number of consecutive iterations in which the best solution must (not)
     have penalties for the penalty value to be updated
3  Input: $\delta_\alpha \leftarrow$ the multiplier for the penalty factor
4  Input: penaltyCounter $\leftarrow$ an integer storing the number of consecutive best
     solutions with or without a penalty
5  if fittest individual has a penalty then
6      if penaltyCounter $< 0$ then
7          penaltyCounter $\leftarrow 0$
8      else
9          penaltyCounter $\leftarrow$ penaltyCounter $+ 1$
10         if penaltyCounter $= \eta_\alpha$ then
11             $\alpha \leftarrow \alpha * \delta_\alpha$
12             penaltyCounter $\leftarrow 0$
13         end
14     end
15 else
16     if penaltyCounter $> 0$ then
17         penaltyCounter $\leftarrow 0$
18     else
19         penaltyCounter $\leftarrow$ penaltyCounter $- 1$
20         if penaltyCounter $= -\eta_\alpha$ then
21             $\alpha \leftarrow \alpha / \delta_\alpha$
22             penaltyCounter $\leftarrow 0$
23         end
24     end
25 end
```

Creating offspring consists of three steps; parent selection, recombination and mutation. The selection of the parents is done with Tournament Selection, which is widely used in the literature according to Miller & Goldberg (1995). This procedure takes *tournamentSize* random chromosomes from the population. The single individual within this tournament with the lowest fitness value, gets selected as a parent. After two tournaments, there are two parent chromosomes selected for recombination. Recombination creates two new chromosomes from these parents. Specifically, for each gene, child one randomly gets the gene from parent one or parent two. Child two gets that gene from the other parent.

To maintain a diverse pool of genes in the population, the chromosomes of the children then get exposed to mutation. Each child has a probability of *mutChance* to be exposed to mutation. If this is the case, each gene of its chromosome has a probability of *mutChanceConditional* to be exchanged for the opposite orientation. We tested this approach and found it to be superior to exposing every gene of every chromosome to a probability of mutation.

5.4 Hybrid Variable Neighbourhood Search and Tabu Search

A VNS/TS is a combination of two metaheuristics, namely a Variable Neighbourhood Search, introduced by Mladenović & Hansen (1997), and a Tabu Search, introduced by Glover (1989). These two metaheuristics have on their own been used for many different problems. Combined they form a strongly diversifying local search, see for example Schneider, Stenger, & Goeke (2014). In our testing this solution technique proved superior to a standalone Tabu Search.

A Variable Neighbourhood Search explores the solution space by iteratively shaking (randomly changing) the solution and then performing a local search around that solution, which is called intensification. The shaking is done based on list of neighbourhoods, sorted from small to large. If no improvement was found during the local search, the next iteration will use the next, larger neighbourhood. In case an improvement was found, the next shaking phase uses the smallest neighbourhood. The local search in the VNS/TS is a Tabu Search. A Tabu Search searches a solution space by performing the best move each iteration, and then declaring that move tabu. Declaring a move tabu means that that part of a solution cannot be changed for a number of iterations. In Algorithm 4, an overview of the VNS of the VNS/TS is provided. The details of the metaheuristics are discussed below.

Algorithm 4: Overview of the VNS of the VNS/TS

```
1  Input: $\mathcal{N} \leftarrow$ a set of neighbourhood structures
2  Input: $\beta_0 \leftarrow$ initial penalty
3  $S \leftarrow ConstrutionHeuristic()$
4  $S \leftarrow TabuSearch(S)$
5  $i \leftarrow 1$
6  $\beta \leftarrow \beta_0$
7  $tabuList \leftarrow$ an empty tabu list
8  while $i <= |\mathcal{N}|$ do
9      $S' \leftarrow Shake(S, \mathcal{N}_i)$
10     Add the flipped edges to $tabuList$
11     $S'' \leftarrow TabuSearch(S', \beta, tabuList)$
12     if $S''$ is an improvement over $S$ then
13         $S \leftarrow S''$
14         $i \leftarrow 1$
15     else
16         $i \leftarrow i + 1$
17     end
18 end
```

The neighbourhoods used by the VNS are based on flipping edges. Each neighbourhood is defined by an integer Λ that represents the number of edges flipped in an iteration. Which edges are flipped, is decided randomly. The flipped edges are also added to the tabu list of the Tabu Search. If Λ is larger than the number of edges in the instance, that neighbourhood is disregarded.

In the intensification phase, a Tabu Search is used. Algorithm 5 provides an overview of this component of the VNS/TS. In the tabu search, the search is performed until no improvement was found for η_{tabu} iterations. The Tabu Search makes use of a tabu list. This list contains tabu attributes, which state that the orientation of a specific edge may not be flipped. The number of iterations the move is tabu, the tabu tenure, is randomly decided on the interval $[\vartheta_{min}, \vartheta_{max}]$. For smaller instances, the tabu list can contain all edges. To reduce this problem, the parameters ϑ_{min} and ϑ_{max} are set to $|V|-7$ and $|V|-1$ respectively if their original values are higher.

Algorithm 5: Overview of the TS of the VNS/TS

```
1  Input: $S \leftarrow$ the solution to improve
2  Input: $\beta \leftarrow$ current penalty factor
3  Input: $tabuList \leftarrow$ list of current tabu attributes
4  Input: $\eta_{tabu} \leftarrow$ the maximum number of iterations without improvement
5  Input: $\vartheta_{min} \leftarrow$ the minimum tabu tenure
6  Input: $\vartheta_{max} \leftarrow$ the maximum tabu tenure
7  Input: $\eta_\beta \leftarrow$ number of consecutive iterations in which the best solution has (not)
     have penalties for the penalty value to be updated
8  Input: $\delta_\beta \leftarrow$ multiplication factor for penalty value
9  $Best \leftarrow S$
10 $noImprovement \leftarrow 0$
11 while $noImprovement < \eta_{tabu}$ do
12     $bestMove \leftarrow$ the best non tabu move from $S$
13     $S \leftarrow$ the solution after performing $bestMove$
14     Decrease all tabu tenures in $tabuList$
15     $tabuList.makeTabu(bestMove, \vartheta_{min}, \vartheta_{max})$
16     if $S$ better then $B$ then
17         $B \leftarrow S$ $noImprovement \leftarrow 0$
18     else
19         $noImprovement \leftarrow noImprovement + 1$
20     end
21     Update $\beta$ using $\eta_\beta$ and $\delta_\beta$
22 end
23 Return: $B$
```

The Tabu Search uses two operators, which we call *EdgeFlip* and *NodeFlip*. The *EdgeFlip* operator flips the orientation of a single edge, and the *NodeFlip* operator flips all edges connected to a particular node. The operator resulting in the best solution which does not contain a tabu move is performed. This procedure may result in moving to a worse solution in an iteration, which leads to the diversification in this search. If there are no non-tabu moves left, a random move is performed.

The search is further improved by allowing the search to make use of the information in the infeasible solution space. This is done with a generalizing cost function similar to the fitness function in the GA. The cost of a solution is the sum of the shortest paths between all feasible pairs plus a penalty β for each infeasible pair. The value of β is initialized at β_0. However, if for η_β consecutive iterations of the Tabu Search the solution has infeasibility penalties, β is multiplied by δ_β. Conversely, if the solution does not encounter a penalty for η_β consecutive Tabu Search iterations, β is divided by δ_β.

6 Results

Multiple computational experiments are performed to asses the performance of the methods. All computations are done on a laptop computer equipped with an Intel Core i7 processor with a base speed of 1.99GHz with 8GB memory, running Windows 10 Home. The code is single threaded and was executed using Eclipse IDE for Java developers version 2019-6 using Java jdk 12. The commercial solver CPLEX Studio 12.10 is used for solving mathematical models. All methods were given a maximum run time of one hour. All tables report the average shortest path distance as the objective. This does not alter the optimal solution compared to that with an objective of minimising the total length of the shortest paths.

This section is structured as follows; In Section 6.1 the chosen parameter settings are discussed. Section 6.2 describes how we generate instances for the EOP. We use these instances in Section 6.3, in which the performance of the construction heuristic is discussed. The performance of a general Benders Decomposition is compared to CPLEX in Section 6.4. Then, we do the same for the specialised Benders Decomposition in Section 6.5. We look at the performances of the Genetic Algorithm and of the VNS/TS in Section 6.6.

6.1 Parameter Tuning

A number of heuristics are implemented for this research. The values for these parameters are decided based on a combination of experience accrued during the development and a grid search. A similar approach is used to decide the neighbourhoods required by the VNS/TS. Table 2 shows the values used by the GA and the VNS/TS. For the Construction Heuristic, *NumberOfCandidates* is set to 5 for the GA, because the GA requires multiple solutions. For the other methods, only one solution is required. In these cases, *NumberOfCandidates* is set equal to 1, such that it is a greedy approach. Moreover, in these methods the edge which is orientated first, normally decided randomly, is always the first edge in the instance. This makes this approach fully deterministic.

Table 2: An overview of the parameters used for the heuristics.

Genetic Algorithm		VNS/TS			
Parameters		Parameters		Neighbourhoods	
$popSize$	500	η_{tabu}	150	k	Λ
$tournamentSize$	20	ϑ_{min}	10	1	5
$mutChance$	0.1	ϑ_{max}	15	2	8
$mutChanceConditional$	0.2	β_0	20	3	11
η_{GA}	25	η_{beta}	4	4	14
α_0	12.5	δ_{beta}	2		
η_α	3				
δ_α	2				

6.2 Data Generation

As this problem is not well-studied, there are no standard test instances available. Therefore, such instances have to be generated. We introduce a procedure that produces simple graphs which have feasible orientations. Algorithm 6 provides an overview of this procedure. We elaborate on the details below.

Algorithm 6: Overview of the data generation process

```
Input: $n \leftarrow$ the number of nodes
Input: $p \leftarrow$ the probability of an extra edge being included
nodes $\leftarrow$ an empty list
edges $\leftarrow$ an empty list
for $n$ iterations do
    Generate node $m$ with its $(x, y)$- coordinates in the interval $[0, 10]$
    Add $m$ to nodes
end
for each node $m$ in nodes do
    while degree$(n) < 3$ do
        Add an edge between $m$ and any other node in nodes to edges
    end
end
while edges and nodes do not form a two-edge-connected graph do
    Add an edge between two under-connected parts to edges
end
for each $m, o$ node pair do
    if there is no edge between $m$ and $o$ then
        With probability $p$, add an edge between $m$ and $o$
    end
end
for each edge $e$ in edges do
    Draw the length of $e$ from a normal distribution
end
```

The procedure takes two parameters, n which represents the number of nodes and p the probability of an extra edge being included. The procedure starts by drawing n (x, y)-coordinates for nodes, such that the coordinates are within the interval $(0, 10)$. Then, for each node, we add edges until the node has a degree of at least three. This may result in nodes having a higher degree.

A degree of three is required, this is due to the following observations; First, a degree of two is necessary as a degree of one will results in an infeasible instance by Robbins' Theorem. Second, a node with a degree of two practically the same as removing that node and combining the two edges into one.

Next, we check whether an instance is two-edge-connected, as then the instance would be feasible by Robbins' theorem. If it is not, edges between under connected parts are added until the instance is connected. Appendix A describes the details of this check. In the next step, the procedure allows for additional edges. Each node pair is considered and if no such edge is included, this edge is added with probability p. The final step is assigning distances to the edges. These are based on normal distribution with the euclidean distance as the mean and that distance divided by $\frac{10}{2}$ as standard deviation. The value 10 chosen as that is the length and width of the plane. If a negative length was decided, the length is altered to zero.

With this procedure, 30 instances are generated. Namely 5 instances for $n = 5$, $n = 10$, $n = 20$, $n = 40$, $n = 70$ and $n = 100$. For each of these, p was set to $1/n$, such that the probability that a node gets an extra edge is constant irrespective of the number of nodes. Table 3 provides an overview of some characteristics of the instances. As a benchmark for our methods, the mathematical model as introduced in Section 3 is implemented in CPLEX.

Table 3: A overview of the characteristics of the different instances.

Instance	5n1	5n2	5n3	5n4	5n5	10n1	10n2	10n3	10n4	10n5
# Nodes	5	5	5	5	5	10	10	10	10	10
# Edges	8	9	9	8	9	19	21	20	21	20
Maximum Degree	4	4	4	4	4	5	5	5	6	5
Instance	20n1	20n2	20n3	20n4	20n5	40n1	40n2	40n3	40n4	40n5
# Nodes	20	20	20	20	20	40	40	40	40	40
# Edges	46	41	43	40	46	84	87	90	80	95
Maximum Degree	7	7	8	6	9	7	7	10	7	10
Instance	70n1	70n2	70n3	70n4	70n5	100n1	100n2	100n3	100n4	100n5
# Nodes	70	70	70	70	70	100	100	100	100	100
# Edges	157	150	153	153	150	217	219	214	219	222
Maximum Degree	9	8	7	8	8	9	8	10	8	8

6.3 Construction Heuristic

In this section, we look at the performance of the construction heuristic. As mentioned before, there are two ways this heuristic is used, namely in a greedy, deterministic fashion and a randomized version which uses a restricted candidate list. For the latter, we performed 100 runs for each instance and report the number of feasible solutions found, the objective of the best found solution, and the average of the objective values of the feasible solutions. We compare the objective values with the best known solutions generated with the other methods. The results are displayed in Table 4. The run time of the construction heuristic is negligible.

Table 4: A overview of the performance of the construction heuristic using the greedy, deterministic settings and 100 runs of the settings with the restricted candidate list. For instances with more nodes, no feasible solution was found.

	Greedy		Restricted Candidate List					
				Best Found		Average		
Instance	Obj. Val.	Gap	# Feasible	Obj. Val.	Gap	Obj. Val	Gap	Best Known
5n1	9.61	5.9%	90	9.08	0.0%	10.06	10.8%	9.08
5n2	9.29	7.0%	100	8.94	3.0%	9.99	15.2%	8.68
5n3	8.14	7.7%	100	8.14	7.7%	9.27	22.6%	7.55
5n4	6.48	7.0%	88	6.48	7.0%	7.31	20.8%	6.05
5n5	9.58	18.1%	100	8.39	3.4%	8.86	9.2%	8.12
10n1	14.05	7.2%	70	13.82	5.5%	14.80	13.0%	13.10
10n2	∞	-	40	10.07	8.2%	10.57	13.6%	9.30
10n3	7.70	16.0%	58	7.21	8.7%	8.00	20.6%	6.63
10n4	8.83	10.7%	81	8.59	7.6%	9.19	15.1%	7.98
10n5	11.59	8.6%	93	11.12	4.1%	12.93	21.1%	10.68
20n1	∞	-	42	13.17	9.4%	14.15	17.5%	12.04
20n2	∞	-	17	18.98	15.3%	20.25	23.0%	16.46
20n3	13.87	14.7%	69	13.40	10.9%	14.46	19.7%	12.09
20n4	16.55	20.1%	33	15.07	9.4%	16.02	16.3%	13.77
20n5	∞	-	6	17.49	37.8%	17.60	38.6%	12.70
40n1	18.73	29.9%	6	18.73	29.9%	19.37	34.3%	14.42
40n2	∞	-	31	17.68	9.7%	18.54	15.0%	16.12
40n3	∞	-	9	14.95	14.8%	15.48	18.8%	13.03
40n4	20.75	30.8%	21	17.97	13.3%	19.28	21.5%	15.86
40n5	∞	-	11	14.23	24.8%	14.69	28.8%	11.41

From the table, we see that the construction heuristic with the restricted candidate list performs finds decent starting solutions. On average, the average is 19.8% away from the best known solution. This figure is 11.5% for the best found solutions. The construction heuristic performs best on smaller instances, finding more feasible solution and objective values closer to the best known solution. For instances with more than 40 nodes, no feasible solutions are found.

Next, we compare the greedy settings for the construction heuristic with the restricted candidate list setting. We see that the greedy settings find a solution for thirteen of the 30 instances. These thirteen solution have on average a 6.2% higher objective value compared to the best of the restricted candidate list. For the larger instances, the construction heuristics often provides infeasible solutions or, for feasible solutions, high objective values.

6.4 General Benders Decomposition

In this section, the EOP instances are solved using both the state of the art solver CPLEX and a general Benders Decomposition. This Benders Decomposition does not use any of the problem specific improvements described in Section 5.2.1, besides the memory improvements. Table 5 displays the results of the instances with up to 40 nodes. The larger instances could not be handles by either of the methods due to memory issues.

Table 5: A comparison of the performance of CPLEX and the general Benders Decomposition. The table displays the run time (RT) in seconds and the upper bound (UB). A dash (-) indicates that no feasible solution could be found. The gap found by the methods and the post-ex gap between the methods is also displayed.

	CPLEX			General Benders Decomposition			
Instance	UB	Gap	RT (s)	UB	Gap	RT (s)	Post-Ex Gap
5n1	9.08		0.3	9.08		1.8	
5n2	8.68		0.1	8.68		2.4	
5n3	7.55		0.0	7.55		2.3	
5n4	6.05		0.1	6.05		1.1	
5n5	8.12		0.1	8.12		2.4	
10n1	13.10		0.3	13.10		1255.9	
10n2	9.30		0.4	9.35	5.8%	3601.7	0.5%
10n3	6.63		0.3	6.63		1764.7	
10n4	7.98		0.3	7.98		3518.7	
10n5	10.68		0.3	10.68		2495.2	
20n1	12.04		335.8	-		3617.6	∞ %
20n2	16.46		124.9	19.56	100.0%	3604.1	18.8%
20n3	12.09		186.0	15.47	100.0%	3623.6	28.0%
20n4	13.77		226.1	18.90	100.0%	3615.2	37.2%
20n5	12.70		82.4	15.54	100.0%	3605.1	22.4%
40n1	14.83	15.1%	3601.2	-			
40n2	16.45	14.0%	3600.7	-			
40n3	13.56	15.5%	3601.3	-			
40n4	22.75	41.0%	3601.1	-			
40n5	11.59	11.2%	3601.4	-			

Overall, we see that CPLEX outperforms the general Benders Decomposition in all regards. Its run time is always shorter and it finds the optimal solution for all instances that the general Benders Decomposition could handle. It can also find solutions for larger instances, those with 40 nodes. However, for these instances it has a large gap, which averages at 19.4%.

There are two main explanations for the poor performance of the general Benders Decomposition. The first is that the optimality cuts are weak. This is because a single cut is generated for the entire solution. Splitting these cuts will better encapsulate the feasible polyhedron, which leads to less cuts being necessary and therefore a lower run time. The second explanation is that the set S is also weak as it includes many infeasible orientations. The consequence is that many iterations can be spent on finding infeasible solutions. This can be seen in instances $20n1$, in which not a single feasible solution was found during the run time of one hour.

6.5 Specialised Benders Decomposition

Many problems encountered by the general Benders Decomposition can be overcome by implementing the problem specific improvements discussed in Section 5.2.1. The resulting specialised Benders Decomposition is again compared to the performance of CPLEX. Table 6 provides an overview of the results.

Table 6: A comparison of the performance of CPLEX and the specialised Benders Decomposition. The table displays the run time (RT) in seconds and the upper bound (UB). A dash (-) indicates that no solution of that type could be found. The gap found by the methods and the post-ex gap between the methods is also displayed.

	CPLEX			Specialised Benders Decomposition			
Instance	UB	Gap	RT (s)	UB	Gap	RT (s)	Post-Ex Gap
5n1	9.08		0.3	9.08		0.2	
5n2	8.68		0.1	8.68		0.2	
5n3	7.55		0.0	7.55		0.1	
5n4	6.05		0.1	6.05		0.1	
5n5	8.12		0.1	8.12		0.2	
10n1	13.10		0.3	13.10		1.0	
10n2	9.30		0.4	9.30		1.4	
10n3	6.63		0.3	6.63		0.8	
10n4	7.98		0.3	7.98		0.9	
10n5	10.68		0.3	10.68		1.0	
20n1	12.04		335.8	12.11	1.8%	3600.0	0.6%
20n2	16.46		124.9	16.46		1056.4	
20n3	12.09		186.0	12.09		1366.3	
20n4	13.77		226.1	13.77	2.0%	3600.0	
20n5	12.70		82.4	12.70		893.7	
40n1	14.83	15.1%	3601.2	15.80	34.3%	3600.0	6.6%
40n2	16.45	14.0%	3600.7	16.56	32.8%	3600.0	0.7%
40n3	13.56	15.5%	3601.3	13.60	29.1%	3600.0	0.3%
40n4	22.75	41.0%	3601.1	16.43	31.3%	3600.0	-27.8%
40n5	11.59	11.2%	3601.4	11.72	23.1%	3600.0	1.1%
70n1				21.41	69.3%	3600.1	
70n2				23.64	58.2%	3600.0	
70n3				20.43	53.7%	3600.0	
70n4				21.17	62.2%	3600.0	
70n5				20.73	52.9%	3600.0	
100n1				23.71	78.5%	3600.1	
100n2				25.82	80.2%	3600.2	
100n3				28.84	81.0%	3600.0	
100n4				26.14	79.6%	3601.6	
100n5				22.84	75.1%	3600.2	

None of the solution techniques in Table 6 is strictly superior to the other. For smaller instances, using CPLEX is faster and provides a smaller optimality gap. For instances with 40 nodes, the upper bound solutions by the Benders Decomposition have a 3.8% lower objective value on average compared to those by CPLEX. However, the gap found by CPLEX for these instances is on average 19.4% which is considerably smaller than the 30.1% by the Benders Decomposition. The main advantage of the specialised Benders Decomposition is that it can handle much larger instances as it uses less memory.

To gain additional insight, instance $40n1$ was run an additional time using the specialised Benders Decomposition while saving the upper bound and lower bound every four minutes. The course of these bounds is plotted in Figure 4. In this graph, it can be seen that the solution technique suffers from a tailing-off effect, i.e. the convergence of the bounds is slower in later iterations. This effect is often observed in Benders Decompositions according to Rahmaniani et al. (2017).

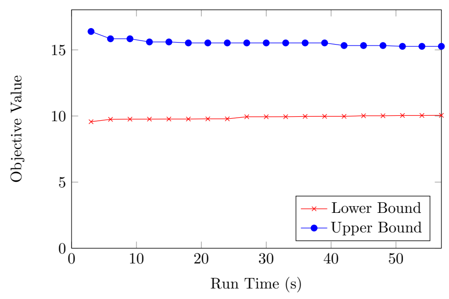


Figure 4: A figure showing the upper bound and the lower bound by the specialised Benders Decomposition every three hours over the course of an hour of solving time. The instance being solved is 40n1.

To gain insight in the determinants of the run time, instance $100n1$ was run an additional time while recording the time spend in the BMP and in the subproblem. The results are in Figure 5. It can be seen that the subproblems is solved fast, making up only 9.9% of the total time spent in both problem combined. The subproblem was solved 173 times and was solved in 2.1 second on average. The relative time use of the BMP goes up the more time goes by. One of the causes for this is that, as more cuts are added, the BMP becomes harder to solve and thus requires more time.

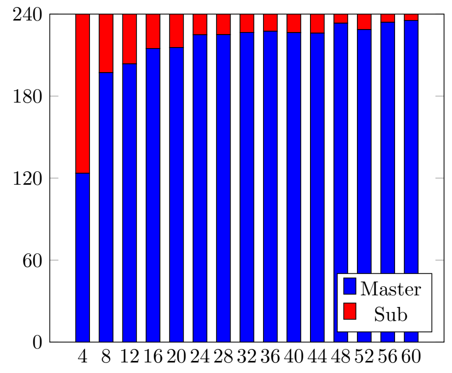


Figure 5: A figure showing the time spent on the master problem and the subproblem for every 4 minutes of run time. The instance being solved is 100n1.

Comparing the performance of the specialised Benders Decomposition in Table 6 with that of the general Benders Decomposition in Table 5, we see that the improvements provide a significant boost in performance. The run times and the gaps found are consistently lower. This is because the reasons for the poor performance of the general Benders Decomposition described in the previous section are reduced. The initial cuts make the set S much stronger. For example, when solving instance $10n5$ the general Benders Decomposition requires 25 feasibility cuts, while the specialised Benders Decomposition requires 0 additional feasibility cuts.

The other reason of the poor performance of the general Benders Decomposition was weak optimality cuts. In the specialised Benders Decomposition these cuts are made and added for each OD-pair separately. This reduces the number of necessary iterations. An added benefit is that solving the subproblem for each OD-pair separately reduces the memory needed. This is what enables the specialised Benders Decomposition to handle much larger instances.

6.6 Metaheuristics

Lastly, we look at the performance of the GA and the VNS/TS-hybrid. Table 7 provides an overview of their performances. In the table, the results are compared to the best solution found by an exact methods, i.e. by CPLEX or one of the Benders Decomposition implementations.

Table 7: An overview of the performances of the GA and the VNS/TS. The table displays the run time (RT) in seconds and the objective value of the best solution found. The gap with the best solution found by one of the exact methods is also displayed.

		Genetic Algorithm			VNS/TS		
Instance	Best Exact	Obj. Val.	RT (s)	Post-Ex Gap	Obj. Val.	RT (s)	Post-Ex Gap
5n1	9.08	9.08	0.2		9.08	0.0	
5n2	8.68	8.68	0.1		8.68	0.0	
5n3	7.55	7.55	0.1		7.55	0.0	
5n4	6.05	6.05	0.1		6.05	0.0	
5n5	8.12	8.12	0.1		8.12	0.0	
10n1	13.10	13.10	0.2		13.10	0.0	
10n2	9.30	9.32	0.2	0.2%	9.30	0.0	
10n3	6.63	6.63	0.2		6.63	0.0	
10n4	7.98	7.98	0.2		7.98	0.0	
10n5	10.68	10.68	0.1		10.68	0.0	
20n1	12.04	12.17	0.3	1.0%	12.07	0.7	0.2%
20n2	16.46	16.80	0.6	2.0%	16.46	1.0	
20n3	12.09	12.25	0.3	1.3%	12.23	0.5	1.2%
20n4	13.77	13.81	0.3	0.3%	13.78	0.5	
20n5	12.70	12.91	0.3	1.7%	12.70	0.9	
40n1	14.83	15.39	1.0	3.8%	14.42	4.1	-2.7%
40n2	16.45	16.43	1.0	-0.1%	16.12	3.8	-2.0%
40n3	13.56	13.52	0.9	-0.2%	13.03	5.7	-3.9%
40n4	16.43	16.59	0.9	1.0%	15.86	4.1	-3.5%
40n5	11.59	11.88	1.0	2.5%	11.41	5.9	-1.6%
70n1	21.41	19.31	4.4	-9.8%	18.30	11.7	-14.6%
70n2	23.64	20.98	4.4	-11.3%	19.95	17.6	-15.6%
70n3	20.43	19.12	5.6	-6.4%	18.13	18.5	-11.3%
70n4	21.17	19.08	4.5	-9.9%	18.50	10.4	-12.6%
70n5	20.73	20.22	4.5	-2.5%	18.85	23.2	-9.1%
100n1	23.71	20.77	17.0	-12.4%	19.34	119.4	-18.4%
100n2	25.82	22.62	15.5	-12.4%	21.29	78.2	-17.6%
100n3	28.84	24.69	16.4	-14.4%	22.70	105.2	-21.3%
100n4	26.14	22.34	16.3	-14.5%	21.09	75.1	-19.3%
100n5	22.84	20.66	15.3	-9.5%	19.20	47.4	-16.0%

The table shows that both metaheuristics are fast. The VNS/TS required on average 12 millisecond to find a solution for the ten smallest instances. For the GA, this figure is slightly higher with 121 milliseconds, however, its run time increases less for larger instances. It found solutions for all 30 instances within two minutes, whereas the VNS/TS took 9 minutes.

Comparing the solutions of the GA to the exact solutions, we see that the GA performs well. Of the first 15 instances, the optimal solution is known and the GA finds this solution for 9 of these. For those instances for which it does not, the gap is 1.1% on average. For the remaining 15 instances of which the optimal solution is not known, the gap is −6.4% on average. In other words, for larger instances the GA finds a better solution in a much shorter time than any of the exact methods.

The VNS/TS consistently outperforms the GA in terms of objective value, as it finds solutions with objective values equal-to or lower-then the objective values of the solutions by the GA. Of the first 15 instances, for which we know the optimal solution, the VNS/TS finds the optimal solution 13 times. For the larger instances, the VNS/TS finds solutions with a gap of −11.3% compared to the best known exact solution. These solutions have, compared to the solutions by the GA, a 5.3% lower objective value.

7 Conclusion

In this thesis, we introduce a new version of the Edge Orientation Problem. This problem considers orientating all edges in a graph such that all nodes can reach each other in such a way that the sum of the distances of all shortest paths is minimised. This problem is rooted in the need for one-way walking paths in the wake of the Covid-19 pandemic, but also has applications to road networks, for example parking garages. Although other EOP versions are considered in the literature, this specific problem is not, to the best of our knowledge.

We contribute to the literature in various way. First, we provide a new proof for the NP-completeness of a related problem. For the EOP. we provide a Mixed Integer Linear Programming formulation for the EOP. Moreover, we provide a procedure to generate problem instances of the EOP and develop four solution methods, each with its own qualities.

We generate 30 instances of the EOP. The commercial solver CPLEX can solve the 15 smallest instances to optimality. It found solutions for the next five instances within an hour with an average optimality gap of 19.4%. CPLEX cannot handle bigger instances due to memory issues. An alternative exact method investigated in this thesis is a Benders Decomposition, of which we implement two versions. The first is a general Benders Decomposition, which does not use problem specific knowledge. This version performs poorly, mainly due to weak cuts. The second version, the specialised Benders Decomposition, does use various pieces of problem specific knowledge. For example, it solves and adds cuts for each OD-pair separately. This version can handle much larger problems than CPLEX, but the gaps it finds remain large.

Additionally, we implement two metaheuristics. The first is a Genetic Algorithm which finds solution for the 30 instances combined within 2 minutes. Of the 15 instances for which the optimal solution is known, the GA finds the optimal solution for 9. For the remaining 6 instances, the gap is 1.1% on average. The second metaheuristic is a hybrid between a Variable Neighbourhood Search and a Tabu Search. This heuristic is slower, requiring 9 minutes to find solutions for the 30 instances, but consistently finds better solutions than the GA. For only two of the 15 instances with a known optimum, this optimum was not found. For the 15 larger instances, the best known solution were found by the VNS/TS. On average, these solutions were 5.3% better then those of the GA. The VNS/TS is therefore the recommended solution technique for large instances in case there is no need for knowledge of the optimality gap.

There are several limitations to this research. First, little effort was put into minimising the run time of the Benders Master Problem. As demonstrated, this makes this approach slow and hampers its performance. One way in which this could be reduced is by applying row management. It is also likely that the subproblem can be sped up. The partial subproblem essentially is the dual of the shortest path problem with additional constraints by the orientation. It is likely that a polynomial time algorithm can be found for this problem. Despite some effort, we were unable to find it. Especially the case for a feasible and bounded partial subproblem proved difficult.

Another limitation is that performances are tested on only 30 instances. This is a relatively low number compared to other research. However, as we test many solutions techniques, an additional instance increases the requires run time considerably.

As there is little research on the EOP, there are plenty of opportunities for further research in various directions. One is to continue the work on the Benders Decomposition and try forms of row management, research algorithms for the partial subproblem, attempt including other forms of feasibility cuts, or implement multi-threading in the subproblem.

Another direction for future research can be to implement other solutions techniques to the EOP as there exists a plethora of metaheuristics and algorithms suitable for the EOP. Moreover, the proposed solution techniques can be adjusted for related problems. For example, the objective function can be changed such that it considers the worst case increase, or such that it considers the relative increase. Other examples, pertaining to the constraints, are allowing a certain number of edges to be without orientations, or by only considering a certain number of OD-pairs, instead of all possible combinations. Lastly, a related and interesting problem for future research is one where the edges have capacities.

References

Arkin, E. M., & Hassin, R. (2002). A note on orientations of mixed graphs. *Discrete Applied Mathematics*, *116*(3), 271–278.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*(1), 238–252.

Boesch, F., & Tindell, R. (1980). Robbins's Theorem for Mixed Multigraphs. *The American Mathematical Monthly*, *87*(9), 716–719.

Chvátal, V., & Thomassen, C. (1978). Distances in Orientations of Graphs. *Journal of Combinatorial Theory, Series B*, *24*(1), 61–75.

CPLEX, IBM ILOG. (2017). *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual Version 12 Release 8.*

Floyd, R. W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM*, *5*(6), 345.

Gamzu, I., Segev, D., & Sharan, R. (2010). Improved Orientations of Physical Networks. In *International Workshop on Algorithms in Bioinformatics* (pp. 215–225).

Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, *1*(3), 190–206.

Hakimi, S. L., Schmeichel, E. F., & Young, N. E. (1997). Orienting graphs to optimize reachability. *Information Processing Letters*, *63*(5), 229–235.

Hassin, R., & Megiddo, N. (1989). On Orientations and Shortest Paths. *Linear Algebra and its Applications*, *114*, 589–602.

Ito, T., Miyamoto, Y., Ono, H., Tamaki, H., & Uehara, R. (2013). Route-Enabling Graph Orientation Problems. *Algorithmica*, *65*(2), 317–338.

Karp, R. M. (1972). *Reducibility Among Combinatorial Problems", Complexity of Computer Computations, eds. RE Miller and JW Thatcher.* Plenum Press.

Miller, B. L., & Goldberg, D. E. (1995). Genetic Algorithms, Tournament Aelection, and the Effects of Noise. *Complex Systems*, *9*(3), 193–212.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Rahmaniani, R., Crainic, T. G., Gendreau, M., & Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, *259*(3), 801–817.

Robbins, H. E. (1939). A Theorem on Graphs, with an Application to a Problem of Traffic Control. *The American Mathematical Monthly*, *46*(5), 281–283.

Schneider, M., Stenger, A., & Goeke, D. (2014). The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations. *Transportation Science*, *48*(4), 500–520.

Silverbush, D., Elberfeld, M., & Sharan, R. (2011). Optimally Orienting Physical Networks. *Journal of Computational Biology*, *18*(11), 1437–1448.

Smith, A. E., Coit, D. W., Baeck, T., Fogel, D., & Michalewicz, Z. (1997). Penalty Functions. *Handbook of Evolutionary Computation*, *97*(1), C5.

A Check for Two-Edge-Connectedness

In our data generation process, we check if the graph is two-edge-connected. This is done, such that we can, by Robbins' theorem, ensure feasibility. This appendix describes how this check is done.

Two-edge-connected means that the graph is still connected after the removal of any edge. As speed is not important for data generation, this is exactly what we do; We remove an edge, check that the graph is still connected, and repeat this process for each edge. We check if a graph is (one-edge-)connected by getting a random node which is added to a set *completed* and adding all its neighbours to a set named *inProcess*. Similarly, each node is, one by one, removed from *inProcess*, added to *completed*, and its neighbours are added to *inProcess*. This process is repeated until the set *inPorcess* is empty. If *completed* contains all nodes, the graph is connected. Otherwise, the graph is not connected. Algorithm 7 provides an overview of this procedure.

Algorithm 7: Overview of the check for two-edge-connectedness

```
1  Input: $G \leftarrow$ the graph to check
2  for each edge $e$ in $G$ do
3      $G' \leftarrow G$ after the removal of $e$
4      completed $\leftarrow$ an empty set of nodes
5      inProcess $\leftarrow$ an empty set of nodes
6      Add a random node from $G'$ to inProcess
7      while $|inProcess| > 0$ do
8          Remove a node $n$ from inProcess
9          for each neighbour $o$ of node $n$ in graph $G'$ do
10             if $o$ not in not in inProcess nor in completed then
11                 add $o$ to inProcess
12             end
13         end
14         Add $n$ to completed
15     end
16     if $|completed| <$ the number of nodes in $G'$ then
17         Return: False
18     end
19 end
20 Return: True
```

In case a graph turns out to not be two-edge-connected, one node is randomly drawn from the set *completed* and another node is randomly drawn from those not in the set *completed*. An edge is constructed between these two nodes. After this, the entire process is performed again, until the graph is two-edge-connected.