ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master's Thesis in Econometrics and Management Science

Operations Research and Quantitative Logistics

The Edge Orientation Problem with Origin-Destination Pairs

Author: Karel Bleichrodt
Student ID Number: 485093

Supervisor
Dr. W. van den Heuvel

Second Assessor
Dr. T. Dollevoet

Date final version: February 21, 2022

Abstract

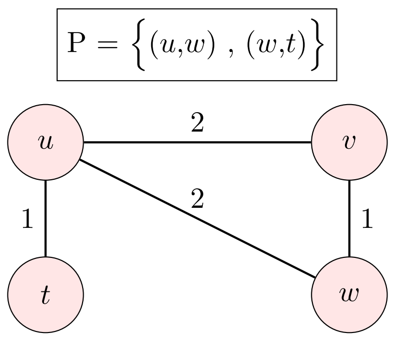
In this thesis, we investigate the *Edge Orientation Problem with Origin-Destination Pairs* (EOP-ODP). The EOP-ODP is associated with an initially undirected graph $G = (V, E)$, where V represents the set of vertices and E represents the set of edges. Each edge has a non-negative length. In addition, the problem involves a set P of Origin-Destination (OD) pairs. Every OD pair contains two vertices, one functioning as its origin and the other functioning as its destination. The objective of the EOP-ODP is to transform each undirected edge into a directed arc, such that a directed path from the origin to the destination exists for each OD pair, and the total distance of the shortest directed OD paths is minimized. The main contributions of this thesis are as follows. We establish that the EOP-ODP can be classified as strongly $\mathcal{NP}$-hard. Next, we develop a polynomial-time algorithm to examine the existence of a feasible solution to any given instance of the EOP-ODP. In the final part of the thesis, several solution approaches are designed. We compare the performance of an MIP formulation, a simple heuristic and an Ant Colony Optimization algorithm. On our generated set of instances, the Ant Colony Optimization algorithm clearly outperforms the other two options.

Contents

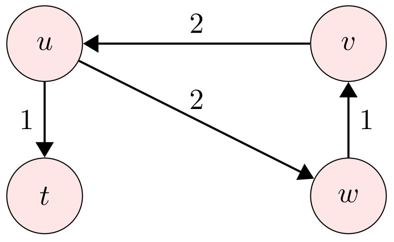
1 Introduction

This thesis investigates the Edge Orientation Problem with Origin-Destination Pairs (EOP-ODP). The EOP-ODP is defined on an initially undirected graph $G = (V, E)$, where V represents the set of vertices and E represents the set of edges connecting pairs of vertices. Throughout this thesis, the amount of vertices is denoted by n and the amount of edges is denoted by m. Each edge of the edge set is accompanied by a non-negative length. The problem contains a set of origin-destination (OD) pairs. This set is denoted by $P = \{(v_1^o, v_1^d), \ldots, (v_h^o, v_h^d)\}$, where $v_1^o, v_1^d, \ldots, v_h^o, v_h^d \in V$, and the total amount of OD pairs is h. The challenge of the EOP-ODP is to transform each undirected edge into a directed arc, such that a directed path from the origin to the destination exists for each OD pair and the total distance of the shortest directed OD paths is minimized.

The EOP-ODP is illustrated graphically in Figure 1. Subfigure 1(a) exhibits an example graph. The graph contains 4 vertices, named t, u, v, w. Furthermore, the example contains two OD pairs. In the first OD pair, vertex u is the origin and vertex w is the destination. The second OD pair has vertex w as origin and vertex t as destination. A possible solution to this example is given in Subfigure 1(b). In this particular solution, the shortest path for the first OD pair has a length of 2, whereas the shortest path for the second OD pair has a length of 4. This implies that the solution is associated with a total objective value of 6.



(a) An example graph for the EOP-ODP.



(b) A possible solution.

Figure 1: Illustration of the EOP-ODP.

The EOP-ODP has numerous practical applications. An important application is the regulation of social-distancing measures that are taken during situations as the current Covid-19 pandemic. In buildings, for example, one wants to avoid people passing each other in opposite directions, due to the risk of contamination. Therefore, a direction is chosen for each hallway within the building. These directions should be chosen in such a way that the additional walking distance, which is an inevitable consequence of the one-direction hallways, is minimized. Other relevant applications emerge in situations where two people or vehicles simply are unable to pass each other (e.g. two nurses pushing large hospital carts in a narrow hallway or narrow streets where vehicles arriving from opposite directions cannot pass).

The remainder of this thesis is organized as follows. In the following section, we provide a literature review. Section 3 is dedicated to the complexity of the EOP-ODP. In Section 4, the conditions for the existence of a feasible solution to the EOP-ODP are derived. Additionally, a procedure to construct a feasible solution is established in this section. In Section 5, our methods are introduced. Then, the construction of the data set is described in Section 6. Section 7 is devoted to evaluating the performance of our methods. Finally, Section 8 contains our conclusions.

2 Literature Review

The question of assigning an orientation to the edges of an undirected graph has not been studied very extensively in the existing literature. W. C.-K. Yen (2006) introduce two separate versions, both of which are slightly different to the Edge Orientation Problem with Origin Destination Pairs (EOP-ODP) that we investigate. In the *Out-Degree Edge-Orientation Problem*, the orientation scheme aims to minimize the maximum out degree among all vertices. The second version is the *Vertex-Weighted Edge-Orientation Problem*, which extends the Out-Degree Edge-Orientation Problem. In the Vertex-Weighted Edge-Orientation Problem, vertices are assigned a fixed cost. The objective is to find an orientation scheme wherein, considering all vertices, the maximum of the sum of (1) the vertex out degree and (2) the vertex cost is minimized. W. C.-K. Yen (2007) extended their previous research by providing results on specific types of graphs. They define an Edge Orientation Problem that contains vertices that are associated with a cost, and edges that are associated with a weight. It is first proven that this Edge Orientation Problem is $\mathcal{NP}$-hard on split graphs (i.e. graphs which can be divided into a clique and an independent set) and planar graphs (i.e. graphs that can be drawn in such a way that the edges solely intersect at the vertex points). Next, algorithms are developed for star graps, trees and simple cactus graphs (i.e. connected graphs for which the edges are part of at most one simple cycle). Yet another variation wherein edges are assigned an orientation was investigated by Chan et al. (2021). The authors aim to transform the edges into arcs, such that the maximum weighted in-degree is minimized. They deliver a $2(1+\epsilon)$-approximation algorithm, that is based on the relationship with the *densest subset problem*. Finally, we have discovered one contribution in the literature that covers the exact same EOP-ODP as us, namely Ito et al. (2013). The authors have investigated three specific types of graphs. First, it is proven that the EOP-ODP is strongly $\mathcal{NP}$-hard for planar graphs. Secondly, they propose polynomial time algorithms for both cycles and cactus graphs.

Another problem that considers the assignment of an orientation to the edges of an undirected graph involves the determination of the *oriented diameter*. The diameter of a graph is the maximum distance between any pair of vertices. The oriented diameter is then defined as the smallest possible diameter given that all edges must be assigned an orientation. One can observe two significant differences with the EOP-ODP. First, all combinations of vertices are taken into account to discover the oriented diameter, while the EOP-ODP specifies a selection of origin-destination (OD) pairs. Second, the oriented diameter amounts to the maximum distance between any two vertices, whereas the EOP-ODP focuses on the total distance of the shortest OD paths. Chvátal & Thomassen (1978) proved that the determination of the oriented diameter is $\mathcal{NP}$-hard. To the best of our knowledge, the vast majority of the research on the topic of finding the oriented diameter is concerned with deriving results on specific types of graphs. Examples include Surmacs (2017), who improve the bound on the oriented diameter of graphs with known minimum degree, and Kwok et al. (2010), who provide an improvement on the existing bound on the oriented diameter of graphs with a diameter of three.

In the EOP-ODP, once all edges have been assigned an orientation, the quality of the solution can not yet be determined trivially. This is because the shortest possible directed OD paths remain to be constructed. Fortunately, efficient algorithms exist for extracting the shortest path from a (directed) graph wherein all edges have non-negative lengths. Dijkstra (1959) provide an algorithm that has a time complexity of $\mathcal{O}(n^2)$, where n denotes the amount of nodes. As Fredman & Tarjan (1987) showed, combining Dijkstra's algorithm with a *Fibonacci heap* can further reduce the complexity to $\mathcal{O}(n \log n + m)$, where m stands for the amount of edges.

Furthermore, it is essential to investigate whether a feasible solution to the EOP-ODP exists. Robbins (1939) derived the condition under which a graph can be oriented in such a way that a directed path exists from any vertex to any other vertex. They prove that it is possible to have directed paths for each arbitrary vertex pair if and only if the initial undirected graph remains connected after the removal of any edge. Note that the existence of a directed path for every vertex pair guarantees a feasible solution to the EOP-ODP, but is not always strictly necessary, as directed paths merely need to be present from the origin to the destination of all OD pairs.

In developing a method that decides the orientation of the edges, it is crucial to possess information on possibly efficient routes for the OD pairs. It might therefore be beneficial to generate a couple of paths per OD pair in the undirected setting as a first step. Note that the previously mentioned Dijkstra's algorithm is not suitable for this purpose, as it is merely capable of deriving a single shortest path for each OD pair. The problem of constructing multiple short paths from a source node s to a sink node t is named the *k-Shortest Path Problem.* Algorithms for this problem will be used as an initialization for one of our methods. An important contribution on the k-Shortest Path Problem was made by J. Y. Yen (1971), who created an exact method. This method sequentially determines the k shortest loopless paths (i.e. the k shortest paths that do not contain cycles). It is based on the observation that, given that the first $k' < k$ shortest paths are known, the next shortest path can be discovered by examining deviations on these k' paths. Different "deviation nodes", nodes from which the deviation of an established path commences, are explored. Vanhove & Fack (2012) developed a heuristic that is based on the above mentioned method of J. Y. Yen (1971). Similar to J. Y. Yen (1971), Vanhove & Fack (2012) focus exclusively on loopless paths, which implies that a node will not appear more than once within a single path. As mentioned, a deviation node is the node from which a new deviating path that is about to be constructed will differ from its original path. This means that the sequence of nodes (from now on referred to as S) that are placed before the deviation node in the original path will remain present at the beginning of the new deviating path. J. Y. Yen (1971) simply forbids the new deviating path to visit any of the nodes in S. Vanhove & Fack (2012) first consider all nodes that are connected to the deviation node. For these nodes, the shortest path to the sink node t is computed. If this shortest path contains any of the nodes in S, causing the deviating path to contain a cycle, it is discarded. In a detailed experimentation on five real-life road networks, Vanhove & Fack (2012) compare their heuristic to multiple exact methods (including the method of J. Y. Yen (1971)). They found that their heuristic is substantially faster than the exact methods, while retrieving paths that are only slightly worse in most cases.

There are also a few contributions on the subject of simultaneously focusing on short and diverse paths. The diversity of k paths depends on the amount of edges that they have in common, a high diversity being equivalent to few edges in common. Chondrogiannis et al. (2015) have designed an algorithm that incorporates path diversity into the question of finding k short paths. The authors obtain the k shortest paths, given that the similarity between each combination of paths is smaller than a pre-specified threshold. In their experimental results, Chondrogiannis et al. (2015) noticed that the running time increases with a larger value of k or a smaller value of the similarity threshold. Voss et al. (2015) propose a heuristic approach to the problem of obtaining short and diverse paths. Certain *obstacles* are simulated. An obstacle is defined as a ball, with a specified radius. The obstacles are inserted at different positions in the graph, and all edges that are covered by an obstacle are invalidated. The authors state that this approach ensures the exploration of different regions of the solution space. It is noted that the heuristic nature of the algorithm, not needing a guarantee of choosing the absolute best paths, stimulates the diversity of the paths, while also securing a low running time.

3 Complexity

This section is dedicated to proving that the EOP-ODP is $\mathcal{NP}$-hard. To demonstrate the $\mathcal{NP}$-hardness of the problem, it is sufficient to prove that the decision version is $\mathcal{NP}$-complete.

Before starting with our proof, we have to note that Ito et al. (2013) have already demonstrated the $\mathcal{NP}$-hardness of the EOP-ODP in a different way than we will do. Regardless, we are of the opinion that our proof has value, as we conceive it to be much less complicated. Our proof was constructed completely independently of Ito et al. (2013), as we were not aware of the existence of this paper at the time.

3.1 Decision version of the EOP-ODP

The decision version of the EOP-ODP investigates the following question. **"Is there a way to orient the edges of the edge set E such that the total distance of the shortest OD paths is at most z?"** Recall that our aim is to prove that this decision version is $\mathcal{NP}$-complete. A first step to proving that the decision version is $\mathcal{NP}$-complete is to show that it is part of the set $\mathcal{NP}$ (i.e. if the answer to the above question is 'Yes', this can be verified in polynomial time). Note that the amount of distinct origin nodes in the set of OD pairs is bounded by n. Given an orientation of the edges, the shortest path from a single origin node to the other nodes can be found in $\mathcal{O}(n^2)$ by Dijkstra's algorithm. We can thus retrieve the total distance of the shortest OD paths within $\mathcal{O}(n^3)$. This implies that the EOP-ODP is in $\mathcal{NP}$.

The second and final step towards proving that the decision version of the EOP-ODP is $\mathcal{NP}$-complete is to provide a polynomial reduction from an established $\mathcal{NP}$-complete problem. The established $\mathcal{NP}$-complete problem that we use for the polynomial reduction is the *3-SAT problem.* The 3-SAT problem contains n_v variables, denoted by $u_1, u_2, \ldots, u_{n_v}$. Variable u_i is associated with literals u_i and $\hat{u}_i$, the latter representing the negation of u_i $(i = 1, 2, \ldots, n_v)$. All variables are assigned the value *true* or *false.* From now on $t(u_i) = T$ denotes the situation wherein variable u_i is true, whereas $t(u_i) = F$ denotes the situation wherein it is false. Note that $t(u_i) = T$ is always accompanied by $t(\hat{u}_i) = F$, and $t(u_i) = F$ is always accompanied by $t(\hat{u}_i) = T$. The 3-SAT problem further consists of n_c clauses, $d_1, d_2, \ldots, d_{n_c}$. Every clause contains at most three literals, and is satisfied if at least one literal is true. The question related to 3-SAT is: **"Is there a truth assignment such that all clauses are satisfied?"**. A polynomial reduction from 3-SAT to the EOP-ODP implies that any arbitrary instance of 3-SAT can be transformed (in polynomial time) into an equivalent instance of the EOP-ODP. The equivalence is present in the fact that the question belonging to the decision version of a 3-SAT instance is answered with 'Yes' if and only if the question belonging to the decision version of the EOP-ODP instance, that was created by transformation, is answered with 'Yes'. This equivalence is exactly what we will demonstrate in the remainder of this section.

3.1.1 Transforming an instance of 3-SAT to EOP-ODP

The procedure of transforming an instance of 3-SAT to an instance of EOP-ODP is described below.

- Create a node for all literals. Literal u_i is represented by node u'_i and literal $\hat{u}_i$ is represented by node $\hat{u}'_i$, for $i = 1, 2, \ldots, n_v$. Each two nodes that are each other's negation are connected by an edge of length 1 (e.g. u'_1 is connected to $\hat{u}'_1$).

- Create a start node named s. This node is connected to all nodes belonging to literals, by an edge of length 1.

- Create a node for each clause. Clause d_j is represented by node d'_j, for $j = 1, 2, \ldots, n_c$. Each of these nodes is connected to the nodes belonging to literals that the clause contains, by edges of length 2. For example, if $d_1 = \{u_1, u_2, \hat{u}_3\}$, then the node d'_1 is exclusively connected to nodes u'_1, u'_2 and $\hat{u}'_3$.

- The following OD pairs are defined:

  1. From u'_i to $\hat{u}'_i$, $i = 1, 2, \ldots, n_v$
  2. From $\hat{u}'_i$ to u'_i , $i = 1, 2, \ldots, n_v$
  3. From s to d'_j , $j = 1, 2, \ldots, n_c$

- The value of z is chosen as $3n_v + 3n_c$.

A visual example of the graph construction is provided in Figure 2. Although this entire proof relies on the 3-SAT problem, the instance in Figure 2 is technically a 2-SAT instance. A 2-SAT instance as example allowed us to create a transformed EOP-ODP that is as easily readable as possible.

$$\boxed{d_1 = \{u_1, u_2\} \qquad d_2 = \{u_1, \hat{u}_2\} \qquad d_3 = \{\hat{u}_1, u_2\}}$$

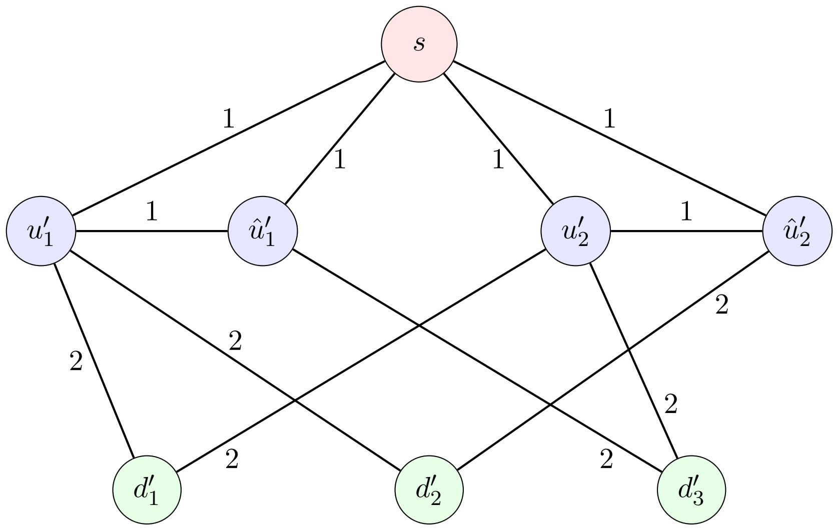


Figure 2: An example of the transformed EOP-ODP graph construction.

The general idea is as follows. The two OD paths between each pair of literal nodes can only be short enough if one arc is directed from s to a literal node and the other arc is directed from the other literal node to s. This means that only one literal node can be reached from s directly, this node represents the "correct" literal, in the 3-SAT truth assignment. Also, the path from s to a clause node (say d'_j) can only be short enough if it directly visits a node (say u'_i) belonging to a "correct" literal that is present in the clause (so $s \rightarrow u'_i \rightarrow d'_j$).

3.1.2 Starting from a 'Yes' instance of 3-SAT

Assume that a truth assignment satisfying all clauses is available for a 3-SAT instance. We have to prove that the instance of the EOP-ODP obtained by the transformation described in Subsection 3.1.1 is also a 'Yes' instance. The orientation of the edges is chosen as follows:

- For example, consider literals u_1 and $\hat{u}_1$, where $t(u_1) = T$ (and therefore $t(\hat{u}_1) = F$) in the given truth assignment. Choose the following arcs:
  1. $s \rightarrow u'_1$
  2. $\hat{u}'_1 \rightarrow s$
  3. $u'_1 \rightarrow \hat{u}'_1$

Now, an OD path from u'_1 to $\hat{u}'_1$ with distance 1 can be constructed. The OD pair from $\hat{u}'_1$ to u'_1 can be covered with a path of distance 2. In the opposite case where $t(u_1) = F$, the roles of u'_1 and $\hat{u}'_1$ should be switched. Continuing this procedure for each pair of literals, one will notice that the total length of the OD pairs between literal nodes is $3n_v$.

- For each literal node belonging to a correct literal in the given truth assignment, one selects arcs (with length 2) from the literal node to all relevant clause nodes. The orientation of the edges (with length 2) connecting the incorrect literal nodes to clause nodes can be chosen arbitrarily. Note that in the previous bullet point, for each pair of literals, it is assured that an arc (with length 1) from s to the correct literal node is present. By assumption, each clause of the truth assignment contains a correct literal. It is thus guaranteed that a path with distance 3 from s to each clause d'_j exists. The total distance of these paths is $3n_c$.
- So, we have found an orientation wherein the total distance of the OD paths is at most $3n_v + 3n_c$, which implies a 'Yes' instance for the EOP-ODP.

Considering the example of Figure 2, the only possible valid truth assignment was $t(u_1) = T$ and $t(u_2) = T$. The corresponding solution to the transformed EOP-ODP is shown in Figure 3. Note that edges going from an incorrect literal node to a clause node are not assigned an orientation yet, as those can be filled in arbitrarily.

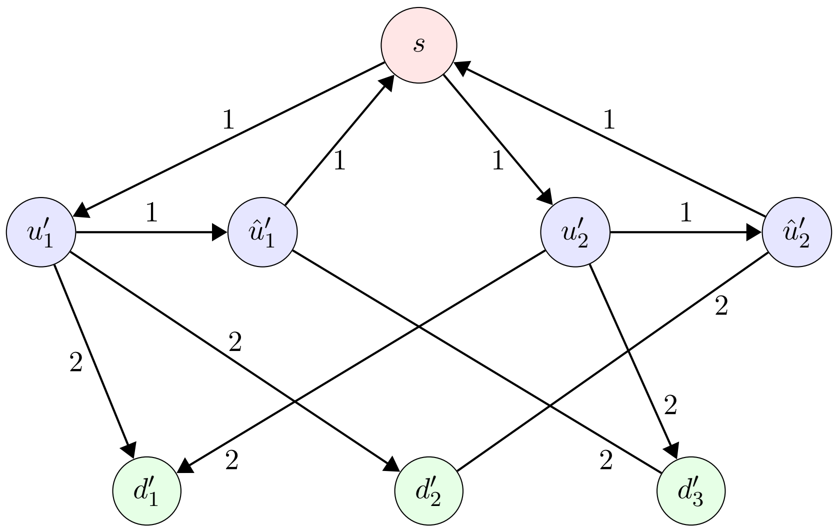


Figure 3: Solution to the example of Figure 2.

3.1.3 Starting from a 'Yes' instance of EOP-ODP

Recall that we created an EOP-ODP instance, by a transformation from a 3-SAT instance, described in Subsection 3.1.1. Assume that we have oriented the edges in this EOP-ODP instance in such a way that the total length of the OD paths is at most $3n_v + 3n_c$. We have to prove that the original 3-SAT instance is also a 'Yes' instance.

- It is easily observed that the total distance of the two OD paths related to each pair of literal nodes is at least 3, as the connecting edge can only be used in one direction. Furthermore, the path of s to an arbitrary clause node d_j has a distance of at least 3. This leads to the conclusion that, in order to satisfy the assumption of a total length not larger than $3n_v + 3n_c$:
    1. The total distance of the two OD paths related to each pair of literal nodes has to be equal to 3. The only way to do this is to form a clockwise or anti-clockwise directed triangle between each pair of literal nodes and s. This means that only one of the nodes belonging to a literal pair can be reached directly from s.
    2. The distance of the OD path from s to each clause node has to be equal to 3. This can only be achieved by directly going from s to a literal node and then directly to a clause node.
- Concluding, for each literal pair, we can select the one literal node that can be reached directly from s in the orientation of the EOP-ODP. Taking the selected literal nodes, it has to be that a connection to all the clause nodes can be made directly, meaning that at least one of the corresponding literals is present in each clause of the original 3-SAT instance. This selected group of literals thus forms a valid truth assignment. In other words, the original 3-SAT instance is a 'Yes' instance.

3.2 Concluding remarks

As we have shown the polynomial reduction from the 3-SAT problem to the EOP-ODP, it can now be concluded that the EOP-ODP is $\mathcal{NP}$-hard. Furthermore, the 3-SAT problem is strongly $\mathcal{NP}$-complete, which implies that the EOP-ODP can be classified as strongly $\mathcal{NP}$-hard. The consequences of this additional finding are that, unless $\mathcal{P} = \mathcal{NP}$, (1) no pseudopolynomial time algorithm exists for the EOP-ODP and (2) no fully polynomial time approximation scheme (FPTAS) exists for the EOP-ODP.

4 Existence and construction of a feasible solution

Before implementing sophisticated methods and algorithms, it is useful to be alarmed when there is no feasible solution to an instance of the EOP-ODP. This scenario occurs when the existence of a directed path for all OD pairs can not be established with assigning a single orientation to the edges of the graph. This section is dedicated to constructing a polynomial time algorithm that provides a certain answer on whether a feasible solution for an instance of the EOP-ODP exists. For the instances of the EOP-ODP that are proven to have a feasible solution, we demonstrate a process of constructing one.

Robbins (1939) provides the definition of an *orientable* graph. An undirected graph is orientable if its edges can be assigned an orientation such that, starting at an arbitrary vertex, a directed path to all other vertices exists. In a graph, *bridges* are defined as edges that, when removed, result in an increase of the amount of connected components. The result that Robbins (1939) derived is stated in Theorem 1.

Theorem 1. *An undirected, connected graph is orientable if and only if it does not contain bridges.*

Now, note that if we have OD pairs for all combinations of two vertices (i.e. having $n^2 - n$ Origin-Destination pairs in total), the graph has to be orientable for a feasible solution of the EOP-ODP to exist. In this special case, we can simply check the amount of bridges in the graph, and draw an appropriate conclusion regarding the possibility of retrieving any feasible solution. However, in the EOP-ODP it is not required that all combinations of two vertices are present in OD pairs. Therefore, the existence of bridges in a graph does not necessarily rule out a feasible solution to the EOP-ODP. In the upcoming part we design a framework which determines whether a feasible solution to the EOP-ODP exists.

Assume that we have an undirected, connected graph $G = (V, E)$. The set P represents the OD pairs. Identify all bridges in G. Identifying all bridges in a graph can be done in $\mathcal{O}(n + m)$, with Tarjan's algorithm. In case these bridges would be removed, the resulting components are $S_1, S_2, \ldots, S_r$. These components are (disjoint) sets of nodes, such that $S_1 \cup S_2 \cup \cdots \cup S_r = V$. Once the bridges are known, the components $S_1, S_2, \ldots, S_r$ can also be found in $\mathcal{O}(n + m)$ time. For each individual component S_i $(1 \leq i \leq r)$, we can assure that its edges can be oriented in such a way that its nodes can be reached from all its other nodes, via a directed path. The reasoning behind this statement is simple. By construction, it is certain that each component S_i does not contain any bridges, as this would have caused the component to be divided into multiple separate components. By Theorem 1, the statement holds. Corollary 1.1 exhibits this result.

Corollary 1.1. *The edges within each component S_i $(1 \leq i \leq r)$ can be oriented such that any node $v \in S_i$ can be reached from $\forall w \in S_i$ $(v \neq w)$.*

Note that the defined components are connected to each other by the bridges of G. If there is a bridge between components S_i and S_j, we denote it by $b(i, j)$ $(1 \leq i \leq j \leq r, i \neq j)$. An example of the division of a graph into components is provided in Figure 4.

Next, we create a *modified graph*, which will capture the original graph G in terms of its components. This modified graph is essential for discovering whether a feasible solution to the EOP-ODP exists. Let the modified graph be denoted by $G' = (V', E')$. Each component S_i $(1 \leq i \leq r)$ in the original graph G is represented by a node in the set V'. Therefore, the nodes in the set V' will be referred to as *component nodes* from now on. The set E' is a set consisting solely of bridges connecting the component nodes in V'. In other words. every bridge in the original graph G is represented by a bridge in the modified graph G'. For the example of Figure 4, the set V' would contain three component nodes and the set E' would contain two bridges. Now, the concept of *Component Origin-Destination pairs* (COD pairs) is presented. The COD pairs are similar to the predefined OD pairs, except for the fact that its origin as well as its destination are component nodes belonging to the set V', instead of nodes in the set V. We construct a Component

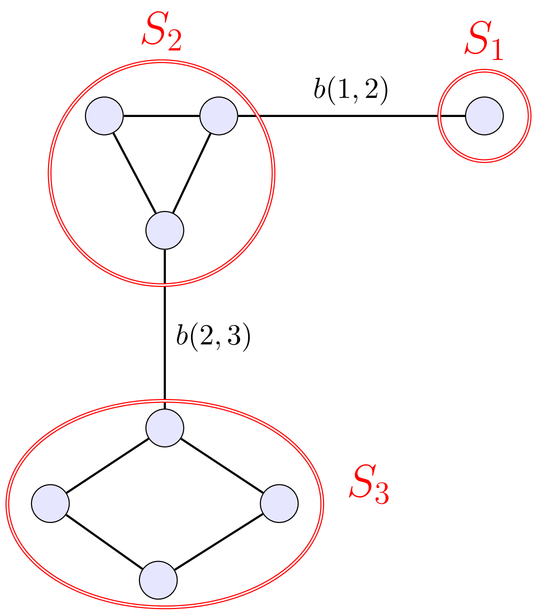


Figure 4: An example, wherein a graph G is divided into components.

Origin-Destination pair from the component node that represents S_i to the component node that represents S_j (with $i \neq j$), when at least one OD pair with an origin node in S_i and a destination node in S_j exists. The condition for a feasible solution of the EOP-ODP is given in Theorem 2.

Theorem 2. *There exists a feasible solution for the EOP-ODP if and only if the bridges of the modified graph G' can be oriented in such a way that a directed path exists for all COD pairs.*

Proof. We divide the proof for this theorem into two parts. Note that if the original graph G does not contain any bridges, no COD pairs are created, and therefore a feasible solution to the EOP-ODP exists.

Proof of Theorem 2: If part

Assume that the bridges of G' have been oriented such that a directed path exists for all COD pairs. To prove that this implies a feasible solution for the EOP-ODP on the original graph G, it has to be shown that an orientation can be chosen for all edges in E, assuring that a directed path exists for all OD pairs in P. A distinction between two types of OD pairs can be made:

- We first consider OD pairs for which the origin node and destination node are in the same component of G. By Corollary 1.1, it is certain that directed paths can be constructed for these OD pairs.
- The other case consists of the OD pairs for which the origin node and destination node belong to different components of G. The edges in E that serve as bridges are assigned the same orientation as their counterparts in the orientation scheme for the bridges of E' in G'.

Consider an arbitrary OD pair with origin node $u_1 \in S_i$ and destination node $u_2 \in S_j$ $(i \neq j)$. Say that (w.l.o.g) the corresponding COD path, expressed in terms of the original components related to the component nodes of V', was $S_i \to S_{i+1} \to S_{i+2} \to \cdots \to S_j$. As Corollary 1.1 indicates, the node in S_i connected to bridge $b(i, i+1)$ can be reached (via a directed path) from u_1. Then, $b(i, i+1)$ can be traversed. From the node in S_{i+1} connected to $b(i, i+1)$, the node in S_{i+1} connected to $b(i+1, i+2)$ can be reached. This procedure is repeated until one arrives at the destination node, constituting a valid directed path for the OD pair.

Proof of Theorem 2: Only-if part

Say that it is not possible to orient the bridges of G' in such a way that a directed path exists for all COD pairs. It is thus not possible to reach the necessary components for every OD pair. As the origin node and destination node of the OD pairs are within components, it is clearly impossible to construct a valid path for all OD pairs. Therefore, no feasible solution for the EOP-ODP exists in G. □

Now that Theorem 2 has been proven, it remains to create an efficient algorithm to examine the existence of a feasible solution for the EOP-ODP. Before introducing the algorithm, an important result is given in Theorem 3.

Theorem 3. *The modified graph G' is a tree.*

Proof. The definition of a *tree* is a connected graph that does not contain any cycles. As the original graph G was connected, it is obvious that G' is also a connected graph. If there were a cycle in G', the edges belonging to this cycle would not be bridges. By construction, every single edge in the edge set E' is a bridge. Therefore, G' does not contain any cycles. This implies that the graph G' can be classified as a *tree.* □

The classification of G' as a tree is a vital aspect for the algorithm to examine the existence of a feasible solution for the EOP-ODP. It is well-established that in a tree, a unique path exists between any pair of nodes. In our case, this means that for every component node there exists a unique path to the other component nodes. This result is exhibited in Corollary 3.1.

Corollary 3.1. *For every component node $v' \in V'$ there exists a unique path to any component node $w' \in V'$ $(v' \neq w')$.*

Finally, the steps that result in obtaining a certain answer on whether a feasible solution to the EOP-ODP exists are provided in Algorithm 1.

Lines 6 and 8 of Algorithm 1 were already described above. According to Corollary 3.1, there is a unique path for each COD pair. This result highly simplifies the procedure that has to be executed. In line 11 of the algorithm, we construct the only path p^* from the origin component node to the destination component node of the current COD pair, using a depth-first search algorithm (e.g. mentioned by Mitchell et al. (1979)). The orientation of the bridges present in p^* is analyzed and

Algorithm 1 The existence of a feasible solution for the EOP-ODP

```
1: **Input:** The graph $G = (V, E)$ plus the set of OD pairs $P$.
2: **Output:** Whether a feasible solution to the EOP-ODP exists.
3: **Complexity:** $\mathcal{O}(n^3)$
4:
5: feasibleSolution = true
6: – Determine the bridges of $G$, and the components $S_1, S_2, \ldots, S_r$.
7: orientation(i,j) = false , for all $1 \leq i, j \leq r$
8: – Create the graph $G' = (V', E')$. Construct a set $Q$ containing all COD pairs.
9:
10: for COD pair $q \in Q$ do
11:    – Construct the only path $p^*$ from the origin component to the destination component of $q$
       (Corollary 3.1).
12:    for Each bridge in $p^*$ from $S_i$ to $S_j$ do
13:       if orientation(j,i) = true then
14:          feasibleSolution = false
15:          BREAK
16:       else
17:          orientation(i,j) = true
18:       end if
19:    end for
20: end for
21: return feasibleSolution
```

chosen accordingly in lines 12-19. If a bridge is given two orientations (indicated by *orientation(i,j)* = *orientation(j,i)* = *true*), then we know that there is no way to orient the bridges of G' such that a directed path exists for all COD pairs. On the other hand, if we can cover all COD pairs without assigning two orientations to any bridge, we have found a way to properly orient the bridges of G'. As Theorem 2 specifies, the information on the possibility of orienting the bridges of G' such that a directed path exists for each COD pair is enough to state whether there is a feasible solution to the EOP-ODP.

Next, we illustrate that Algorithm 1 is executed in $\mathcal{O}(n^3)$ time. Recall that the amount of OD pairs is denoted by h. For line 6, retrieving all bridges in a graph can be done in $\mathcal{O}(n + m)$ time, with Tarjan's algorithm. Once the bridges are known, the components $S_1, S_2, \ldots, S_r$ can also be found in $\mathcal{O}(n + m)$ time. In line 8, the construction of the graph G' then takes $\mathcal{O}(r + m)$ time. Constructing the COD pairs in line 8 takes $\mathcal{O}(h)$ time. For lines 10-20, the complexity is as follows. The amount of COD pairs (line 10) is bounded by h. In line 11, to find the unique path p^* in G' for a specific COD pair, a depth-first search algorithm can be used. As Mitchell et al. (1979) indicate, there exists a depth-first search algorithm with complexity $\mathcal{O}(r)$ for this particular situation of finding the unique path between vertices in a tree. The depth-first search algorithm is invoked for each COD pair. Furthermore, the amount of bridges in p^* (line 12) is bounded by r. This leads to the conclusion that lines 10-20 can be executed in $\mathcal{O}(h \cdot r)$ time. Note that (1) the amount of components r is never greater than n and (2) both the amount of edges m and the amount of OD pairs h are bounded by n^2. Therefore, the total time complexity of the algorithm is $\mathcal{O}(n^3)$.

If Algorithm 1 shows that a feasible solution to the EOP-ODP exists, it is also a suitable basis for constructing one. First, the bridges connecting the components $S_1, S_2, \ldots, S_r$ must be oriented according to the *orientation(i,j)*, which results from Algorithm 1. Then, Atallah (1984) mentions a procedure that can be utilized for the other edges. They briefly describe a manner to orient the edges of any undirected, connected and bridgeless graph such that a directed path exists between each pair of vertices. The idea is to (1) find a depth-first spanning tree, (2) orient its edges in the parent-to-child direction and (3) orient the remaining non-tree edges from the descendant vertex to the ancestor vertex. This procedure can be applied to each individual component (recall that each component was shown to be connected and bridgeless). The complexity of applying this procedure to all components is $\mathcal{O}(m)$. Therefore, we can conclude that this way of constructing a feasible solution to the EOP-ODP likewise has a total complexity of $\mathcal{O}(n^3)$.

5 Methodology

Before introducing our methods, we define some notation, a part of which was already defined in previous sections. The EOP-ODP is defined on an initially undirected graph $G = (V, E)$. The set V represents the set of nodes, whereas E represents the set of edges. The amount of vertices is denoted by n, and the amount of edges is denoted by m. Every edge $e \in E$ is associated with a non-negative edge length c_e. For the methods that are proposed in this section, it is more convenient to work with an arc set A. The arc set A is formed by turning every edge into two arcs (one for each direction), such that $|A| = 2 \times |E|$. Let (i, j) and $(j, i) \in A$ represent the two arcs corresponding to edge $e = (i, j) \in E$, then the length of the arcs is specified as $c_{ij} = c_{ji} = c_e$. The resulting graph is denoted by $H = (V, A)$. Last, the problem contains a set P of origin-destination (OD) pairs, $P = \{(v_1^o, v_1^d), \ldots, (v_h^o, v_h^d)\}$. Here, $v_1^o, v_1^d, \ldots, v_h^o, v_h^d \in V$ and h represents the total amount of OD pairs. For OD-pair $p \in P$, the origin node is denoted by $O(p)$ and the destination node is denoted by $D(p)$.

5.1 MIP formulation

An MIP formulation of the EOP-ODP is provided in this subsection. The MIP formulation is expected to be able to solve small-size instances to optimality. However, regarding the (strong) $\mathcal{NP}$-hardness that was proven in Section 3, the MIP formulation is unlikely to perform well on larger instances. Our MIP formulation of the EOP-ODP extends the MIP formulation for the Shortest-Path Problem that was stated by Taccari (2016). Additional to the notation that was defined at the beginning of Section 5, let $\delta^+(i)$ denote the set of outgoing arcs for node $i \in V$. Conversely, $\delta^-(i)$ denotes the set of incoming arcs for node $i \in V$. A binary variable X_{ij}^p is defined for each arc $(i, j) \in A$ and each OD pair $p \in P$. If the shortest path for p contains arc (i, j), then X_{ij}^p will take a value of 1. Otherwise, X_{ij}^p is equal to zero. The binary variable Y_{ij}, defined for each arc $(i, j) \in A$, is associated with a value of 1 if and only if the arc (i, j) is enabled in the orientation scheme. This leads to the following formulation.

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} \sum_{p\in P} c_{ij} \cdot X_{ij}^{p} \\
\text{s.t.} \quad & \sum_{(i,j)\in\delta^{+}(i)} X_{ij}^{p} - \sum_{(j,i)\in\delta^{-}(i)} X_{ji}^{p} = \begin{cases} 1, \text{ if } i = O(p) \\ -1, \text{ if } i = D(p) \\ 0, \text{ otherwise} \end{cases} && \forall i \in V,\, \forall p \in P \\
& X_{ij}^{p} \leq Y_{ij} && \forall (i,j) \in A,\, \forall p \in P \\
& Y_{ij} + Y_{ji} = 1 && \forall (i,j) \in E \\
& X_{ij}^{p},\, Y_{ij} \in \{0,1\} && \forall (i,j) \in A,\, \forall p \in P
\end{aligned}
$$

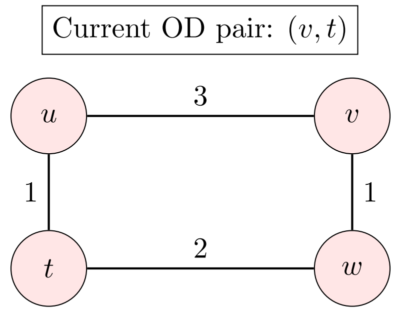
The objective function minimizes the distance of the shortest OD pairs. The first constraint regulates the flow related to all nodes $i \in V$, for all OD pairs $p \in P$. The second constraint ensures that arcs that are not enabled in the orientation scheme cannot be used to form a path for any OD pair. The third constraint imposes the condition that, given two arcs that connect the same pair of nodes in opposite directions, only one can be selected. The last constraint limits the decision variables to binary values.

5.2 Simple heuristic

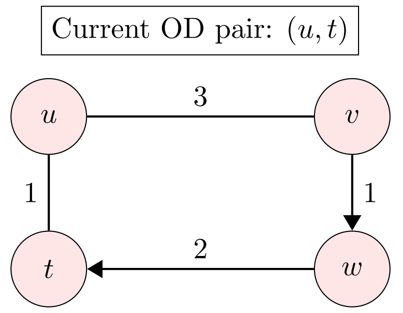
In this part, a simple heuristic is composed. The simple heuristic performs numerous iterations. The iterations operate independently. In every iteration, the aim is to construct a feasible, as-good-as-possible solution. The procedure that is executed within each iteration is as follows. An initially empty list L of prohibited arcs is maintained. The OD pairs are considered sequentially, in random order. For each OD pair, Dijkstra's algorithm is used to create a shortest path, avoiding the arcs that are present in L. The arcs of the created shortest path are used, meaning that all corresponding opposite arcs are added to L. The process of (1) finding the shortest path on a graph in which the arcs in L are forbidden and (2) consequently updating the list L, is repeated until all OD pairs are covered. It is possible that we encounter an OD pair for which, due to certain arcs being added to L, no directed path can be constructed. In this case, the iteration is terminated immediately. At the end of each iteration, the solution is evaluated. If the iteration was terminated prematurely, there is no need to evaluate the solution, as it is guaranteed to be infeasible. After evaluating the solution (regardless of whether the solution was feasible), all elements are removed from L, indicating that every arc is possible again at the start of the next iteration. The final objective value provided by the simple heuristic corresponds to the best solution value among all iterations. As stopping criterion, a total time limit of t^{SH} is imposed.

The process of an iteration of the simple heuristic is illustrated with an example in Figure 5.

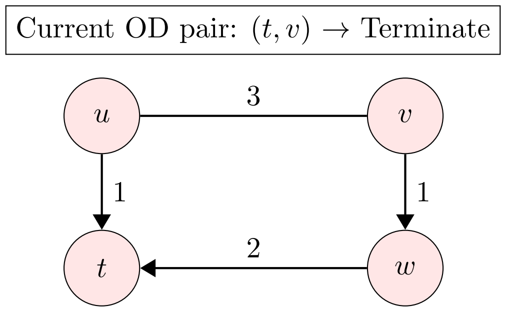
In Figure 5, we are considering a graph with four vertices (named t, u, v, w), and a couple of edges connecting these vertices. Recall that the simple heuristic maintains a list L of prohibited arcs and covers the OD pairs sequentially, in a random order. In attempt to stimulate the conciseness



(a) Currently considering OD pair (v, t).



(b) Currently considering OD pair (u, t).



(c) Currently considering OD pair (t, v). Terminate the simple heuristic.

Figure 5: An example iteration of the simple heuristic.

of Figure 5, the list L is not explicitly shown in mathematical notation. In Subfigure 5(a), the OD pair (v, t) is encountered. According to the procedure of the Simple Heuristic, the shortest path is found. The arcs of the shortest path are fixed, the corresponding opposite arcs are prohibited, and we move on to Subfigure 5(b) for the next OD pair (u, t). The same procedure, while avoiding all prohibited arcs, is executed for this OD pair. Then, we arrive at Subfigure 5(c). Here, the current OD pair is (t, v). When analyzing the graph of this subfigure, one can observe that it has been rendered impossible to construct a path for this OD pair. Therefore, this particular iteration of the simple heuristic is subsequently terminated.

5.3 Ant Colony Optimization algorithm

Ant Colony Optimization algorithms are inspired by the real-world behaviour of ants. Ants following a trail are known to leave a substance called *pheromone*. Pheromone is appealing to ants, causing a higher pheromone level on a trail to attract more ants. This can be translated to the EOP-ODP by assigning a pheromone level to solution components. At first sight, it appears to be logical to assign a pheromone level to each arc in the problem. Arcs with a high pheromone level would then be likely to be enabled in the orientation scheme. However, in preliminary experiments, the results delivered by this approach were not very promising.

Instead of the arcs of the EOP-ODP, we opted for the directed paths from the origin to destination of the OD pairs as solution components. For each OD pair, numerous directed paths from the origin to the destination are assigned an, initially equal, pheromone level τ^0. In a solution to the EOP-ODP, exactly one path must be selected for each OD pair. A total of N ants is present, every ant constructs a solution in each iteration. In the process of constructing a solution, the pheromone level of the paths is taken into account. A path that has a high pheromone level will have a high probability of being chosen. After constructing the solution, the quality of the solution is evaluated. If the solution is of high quality, the pheromone levels of the paths that are present in the solution are increased. The process of (1) constructing solutions (partially) based on pheromone level, (2) evaluating (and improving) the solutions and (3) updating the pheromone levels of the paths accordingly is constantly repeated. Eventually, the paths that are present in many good solutions will be associated with a high pheromone level, and therefore will be chosen more often, directing the algorithm towards high-quality solutions. The outline of the Ant Colony Optimization algorithm for the EOP-ODP is presented in Algorithm 2.

5.3.1 Step 0: Generating paths

In this step, k paths on the graph $H = (V, A)$ are generated for each OD pair. Recall that H is essentially equivalent to the original graph $G = (V, E)$, as two arcs are created for every edge $e \in E$. A path is defined as an ordered list of arcs: $\left\{(v_{i_0}, v_{i_1}), (v_{i_1}, v_{i_2}), \dots, (v_{i_{b-1}}, v_{i_b})\right\}$, where b denotes the number of arcs in the path. The information of k paths on graph H assists in eventually making smarter decisions on the arcs that are enabled. Logically, a key aim is to generate short paths. We

Algorithm 2 Ant Colony Optimization algorithm

```
Input: The graph $H = (V, A)$, with arc lengths $c_{ij}$ for each $(i, j) \in A$. The set $P$, consisting of all OD pairs.
Output: The best discovered solution $s^{gb}$.

Step 0: Generate $k$ paths per OD pair.
while stopping criterion not reached do
  for ant $n = 1, 2, \ldots, N$ do
    Step 1: Create a solution partially based on path pheromone level.
    Step 2: Improve the solution. Update the best solution $s^{gb}$.
  end for
  Step 3: Update the pheromone level of the paths.
end while
```

use the simple heuristic presented by Vanhove & Fack (2012) to generate the k paths. The paths that are connected to OD pair $p \in P$ are stored in the set $W(p)$.

It has to be noted that a substantial amount of algorithms related to selecting k short paths have been developed. Finding the shortest k paths can be done using exact methods to the k-Shortest Path Problem. However, Vanhove & Fack (2012) have compared their heuristic to multiple state-of-the-art exact methods, revealing indications that their heuristic is much faster, at the cost of delivering slightly sub-optimal paths. As Step 0 merely serves as an initialization for the Ant Colony Optimization algorithm, and k paths have to be generated for each and every OD pair, a fast approach is certainly desirable. There exist algorithms that aim to construct k paths that are scored on length as well as diversity (e.g. Chondrogiannis et al. (2015) and Voss et al. (2015)). The diversity of k paths is high when the paths have a small amount of edges in common. It is conceivable that this approach is advantageous for the EOP-ODP. When the k paths largely contain the same edges, it happens more frequently that only a small amount of paths remain possible, as a result of an arc being fixed in an unfortunate direction. Although accounting for diversity of the paths could thus certainly have advantages for the EOP-ODP, the speed advantage of the heuristic of Vanhove & Fack (2012) was considered more essential.

5.3.2 Step 1: Constructing solutions

Step 1 constitutes a construction heuristic. In this step, the OD pairs are handled sequentially, in a random order. The procedure that is applied for each OD pair $p \in P$ is to (1) select a suitable path among the paths in $W(p)$, based on a score function and (2) fix the arcs of the selected path. The score function, which aims to capture the attractiveness of each individual path in $W(p)$, consists of two components. The first component is the pheromone level associated with a path, which is denoted by τ_w for every $w \in W(p)$. The length of the paths comprises the second component. Recall that the length of an arc $(i, j) \in A$ is represented by c_{ij}. For each $p \in P$, the score belonging to a path $w \in W(p)$ is given by

$$S(w,p) = \frac{\Big(\tau_w\Big)^{\alpha}}{\Big(\sum\limits_{(i,j)\in w} c_{ij}\Big)^{\beta}} \quad , \tag{1}$$

where α and β represent positive-valued parameters. One can clearly observe that a higher pheromone level of a path and a lower length of a path result in a higher score. Now, the task remains to select a path based on the scores. Note that not all k paths from the set $W(p)$ belonging to $p \in P$ are necessarily feasible. Every path that contains an arc $(i,j) \in A$ for which the opposite arc $(j,i) \in A$ has already been fixed, should be eliminated. The set of paths that remain feasible is denoted by $W'(p)$. The probability of selecting path w from $W'(p)$ is denoted by

$$Pr(w, W'(p)) = \frac{S(w,p)}{\sum\limits_{u\in W'(p)} S(u,p)} \quad . \tag{2}$$

All arcs of the selected path will be fixed.

In an unfortunate scenario, none of the paths in $W(p)$ is feasible, causing $W'(p)$ to be empty. In other words, each of the paths that were constructed for OD pair p has been turned infeasible by the current fixation of certain arcs in the graph. However, it is still possible for a different valid path to exist. Therefore, Dijkstra's algorithm is applied on the current graph. If a valid path is found, the corresponding arcs are fixed. Additionally, this path is added to $W(p)$ to provide a wider range of paths in future iterations, also attaching a dynamic aspect to $W(p)$. The initial pheromone level that is assigned to the retrieved path is equal to the average pheromone level of the paths in $W(p)$. If no valid path is found, it is no longer possible to find a feasible solution, hence the construction heuristic is terminated. Algorithm 3 provides an exhibition of Step 1.

5.3.3 Step 2: Improving the solutions

We have demonstrated the way in which a total of N solutions are created in Step 1 of the Ant Colony Optimization algorithm. For each individual solution, certain arcs were fixed (implying that other arcs were forbidden), and a single path was selected for the OD pairs.

In Step 2, the aim is to improve the solutions of Step 1. The point for improvement is revealed by observing that the OD paths in Step 1 are selected according to their score function, with a random component. Due to (1) the presence of pheromone level in the score function and (2) the fact that the path with the highest score is not always picked, it is possible that certain paths that were selected in Step 1 are not the shortest paths possible. Therefore, the approach of Step 2 is to invoke Dijkstra's algorithm again, to determine the actual shortest paths for each OD pair on the directed graph. Note that this improvement step is only applied to solutions for which the construction heuristic in Step 1 was not interrupted, as an interruption of the construction heuristic is permanently accompanied by an infeasible solution.

Algorithm 3 Step 1 - Ant Colony Optimization algorithm

```
for every OD pair $p \in P$ do
  → Create $W'(p)$, by eliminating the infeasible paths from $W(p)$.

  if $W'(p)$ is non-empty then
    → Select a single path $w \in W'(p)$ with probability $Pr(w, W'(p))$

    → Fix the arcs of the selected path.

  else if $W'(p)$ is empty then
    → Aim to construct an alternative path $p'$ using Dijkstra's algorithm.
    if $p'$ could be constructed then
      → Fix the arcs of $p'$.
      → Add $p'$ to $W(p)$.
    else
      → BREAK out of for loop.
    end if
  end if
end for
```

5.3.4 Step 3: Updating pheromone level

Finally, the pheromone level of the paths is updated in Step 3. Blum (2005) have provided a survey that includes different ways of updating the pheromone level of solution components in Ant Colony Optimization algorithms. They indicated the *Max-Min Ant System* (MMAS), introduced by Stützle & Hoos (2000), as "one of the most successful ACO variants today". Therefore, we have decided to construct a pheromone level updating framework that resembles the MMAS. The most distinguishing features of the MMAS are (1) the usage of merely the best solutions to update the pheromone level and (2) the imposition of bounds on the pheromone level of the solution components, in order to avoid the undesired scenario of certain solution components excessively dominating the construction process. The first point is incorporated in our ACO algorithm for the EOP-ODP as follows. Let s^{ib} denote the best discovered solution among N attempts in the current iteration and s^{gb} denote the currently best discovered solution in the entire algorithm (i.e. the global-best solution). As the solution components in the ACO algorithm are paths for the OD pairs, both s^{ib} and s^{gb} consist of a total of h OD paths. The solution values corresponding to the solutions are denoted by $f(s^{ib})$ and $f(s^{gb})$ respectively. In most iterations, the paths that belong to s^{ib} are granted an addition in pheromone level. The pheromone level update belonging to this case is exhibited as follows:

$$\tau_w \leftarrow \rho \cdot \tau_w \ + \ \Gamma[w, s^{ib}] \cdot \frac{1}{f(s^{ib})} \qquad , \forall w \in W(p), \forall p \in P \,. \tag{3}$$

Here, ρ is a parameter that controls the fraction of the pheromone of the paths that is automatically transferred to the next iteration. An important aspect related to the value of ρ is the speed at which the pheromone level of paths that have not recently been part of the best solution decreases. The lower the value of ρ, the faster these "unsuccessful" paths are essentially discarded due to obtaining an extremely low pheromone level. When quickly (essentially) discarding these paths, one stimulates the use of promising paths, at the risk of permanently ignoring certain paths that might have turned out to be promising later. Therefore, higher values of ρ are typically associated with a better exploration of the solution space and a larger amount of time before reaching high-quality solutions. Furthermore, in Equation (3), $\Gamma[w, s^{ib}]$ is an operator that returns 1 if a path w is part of the solution s^{ib} and zero otherwise. Once every π iterations, in order to stimulate the features of the global-best solution s^{gb}, this solution is utilized for the pheromone level update of the paths (instead of using s^{ib}). This pheromone level update is identical to Equation (3), except for the fact that s^{ib} is replaced by s^{gb}.

Recall that the second distinguishing feature of the MMAS is the bounds that are imposed on the pheromone level of the solution components. Following this approach, we introduce a maximum pheromone level τ_{max} for the OD paths. In case the pheromone level of a certain path reaches a value higher than τ_{max}, the pheromone level is immediately decreased to τ_{max}. The upper bound on the pheromone level is a measure taken to avoid a scenario wherein certain OD paths excessively dominate the solution construction process in Step 1 of the Ant Colony Optimization algorithm.

5.4 Lower bound

So far, the current Section 5 has dealt with algorithms that aim to deliver as-good-as-possible solutions to the EOP-ODP. To properly evaluate these solutions, methods that provide a tight lower bound are desirable. A good benchmark for methods focussing on lower bounds is the value of the LP relaxation of the MIP formulation. Unfortunately, we have not succeeded in developing a method that delivers a tighter lower bound than the LP relaxation. Although our attempts did not result in a well-performing technique, we have established with certainty that a particular Lagrangian relaxation of the MIP formulation for the EOP-ODP always leads to the same lower bound as the LP relaxation. This is exhibited in the remainder of the subsection.

Lagrangian relaxation

We will relax the first set of constraints in the MIP formulation of Subsection 5.1. Three different scenarios are present.

1. $i = O(p)$
2. $i = D(p)$
3. $i \neq O(p)$ and $i \neq D(p)$

The Lagrange multiplier $\lambda_i^p \in \mathbb{R}$ is introduced for each $i \in V$ and each $p \in P$. This yields the following objective function, wherein the three scenarios are incorporated respectively:

$$\sum_{(i,j)\in A}\sum_{p\in P} c_{ij}\cdot X_{ij}^p + \sum_{i\in V}\Bigg[\sum_{\substack{p\in P:\\ i=O(p)}} \lambda_i^p\big(-1+\sum_{\substack{(i,j)\in\\ \delta^+(i)}} X_{ij}^p - \sum_{\substack{(j,i)\in\\ \delta^-(i)}} X_{ji}^p\big) \;+$$

$$\sum_{\substack{p\in P:\\ i=D(p)}} \lambda_i^p\big(1+\sum_{\substack{(i,j)\in\\ \delta^+(i)}} X_{ij}^p - \sum_{\substack{(j,i)\in\\ \delta^-(i)}} X_{ji}^p\big) \;+$$

$$\sum_{\substack{p\in P:\\ i\neq O(p),\\ i\neq D(p)}} \lambda_i^p\big(\sum_{\substack{(i,j)\in\\ \delta^+(i)}} X_{ij}^p - \sum_{\substack{(j,i)\in\\ \delta^-(i)}} X_{ji}^p\big)\Bigg].$$

Some constant Lagrange multiplier terms of the objective function can be put apart. Once this is done, the terms belonging to the three different scenarios can be merged into a single summation. This results in:

$$\sum_{i\in V}\Bigg[\sum_{\substack{p\in P:\\ i=O(p)}} -\lambda_i^p + \sum_{\substack{p\in P:\\ i=D(p)}} \lambda_i^p\Bigg] \;+$$

$$\sum_{(i,j)\in A}\sum_{p\in P} c_{ij}\cdot X_{ij}^p \;+\; \sum_{i\in V}\Bigg[\sum_{p\in P}\lambda_i^p\big(\sum_{\substack{(i,j)\in\\ \delta^+(i)}} X_{ij}^p - \sum_{\substack{(j,i)\in\\ \delta^-(i)}} X_{ji}^p\big)\Bigg].$$

The constant Lagrange multiplier terms will be temporarily omitted from the derivations, in order to create more compact and accessible equations. The remaining terms can be rewritten in the following way, when being grouped according to $\delta^+(i)$ and $\delta^-(i)$:

$$\sum_{(i,j)\in A}\sum_{p\in P} c_{ij}\cdot X_{ij}^p \;+$$

$$\sum_{i\in V}\Bigg[\sum_{\substack{(i,j)\in\\ \delta^+(i)}}\sum_{p\in P}\lambda_i^p X_{ij}^p\Bigg] \;-\; \sum_{i\in V}\Bigg[\sum_{\substack{(j,i)\in\\ \delta^-(i)}}\sum_{p\in P}\lambda_i^p X_{ji}^p\Bigg].$$

Now, we notice that $\sum_{i\in V}\sum_{(i,j)\in\delta^+(i)}$ corresponds to considering all outgoing arcs, for all nodes $i\in V$. This is equivalent to considering all arcs exactly once, which is $\sum_{(i,j)\in A}$. Using the same sort of logic, $\sum_{i\in V}\sum_{(j,i)\in\delta^-(i)}$ is also equivalent to $\sum_{(i,j)\in A}$. Therefore, we can rewrite the equations as:

$$\sum_{(i,j)\in A}\Bigg[\sum_{p\in P}\big(\lambda_i^p + c_{ij}\big)X_{ij}^p\Bigg] \;-\; \sum_{(i,j)\in A}\Bigg[\sum_{p\in P}\lambda_i^p X_{ji}^p\Bigg].$$

Combining these two terms, and reinserting the constant terms that were omitted temporarily, we obtain the following final objective function:

$$\sum_{i\in V}\Bigg[\sum_{\substack{p\in P:\\ i=O(p)}} -\lambda_i^p + \sum_{\substack{p\in P:\\ i=D(p)}} \lambda_i^p\Bigg] \;+ \sum_{(i,j)\in A}\Bigg[\sum_{p\in P}\Big(\big(\lambda_i^p + c_{ij}\big)X_{ij}^p \;-\; \lambda_i^p X_{ji}^p\Big)\Bigg].$$

The non-constant terms of this final objective function are clearly separable in each arc. As we have not relaxed all constraints, the non-relaxed constraints remain to be satisfied. The feasible region is defined by:

$$\begin{aligned} F^R = \quad & \Big\{(\mathbf{X},\mathbf{Y}) : X_{ij}^p \le Y_{ij} \;\;,\;\; \forall (i,j)\in A\,,\, \forall p \in P \;\;; \\ & \qquad Y_{ij} + Y_{ji} = 1 \;\;,\;\; \forall (i,j) \in E \;\;; \\ & \qquad X_{ij}^p \in \{0,1\} \;\;,\;\; \forall (i,j)\in A\,,\, \forall p \in P \;\;; \\ & \qquad Y_{ij} \in \{0,1\} \;\;,\;\; \forall (i,j) \in A \Big\}. \end{aligned}$$

The problem of solving the Lagrangian relaxation given fixed Lagrange multipliers will be abbreviated by $LR(\lambda)$ from now on. Its definition is obtained by combining the final objective function and the feasible region F^R:

$$\begin{aligned} LR(\lambda) \;\; = \;\; & \sum_{i \in V} \Big[\sum_{\substack{p\in P:\\ i=O(p)}} -\lambda_i^p + \sum_{\substack{p\in P:\\ i=D(p)}} \lambda_i^p \Big] \;\; + \\ & \min_{X,Y} \;\Big\{ \sum_{(i,j)\in A} \Big[\sum_{p \in P} \Big((\lambda_i^p + c_{ij}) X_{ij}^p \;-\; \lambda_i^p X_{ji}^p \Big) \Big] \quad : \\ & \qquad X_{ij}^p \le Y_{ij} \;\;,\;\; \forall (i,j)\in A\,,\, \forall p \in P \;\;; \\ & \qquad Y_{ij} + Y_{ji} = 1 \;\;,\;\; \forall (i,j) \in E \;\;; \\ & \qquad X_{ij}^p \in \{0,1\} \;\;,\;\; \forall (i,j)\in A\,,\, \forall p \in P \;\;; \\ & \qquad Y_{ij} \in \{0,1\} \;\;,\;\; \forall (i,j) \in A \Big\}. \end{aligned}$$

Note that arcs (i,j) and (j,i) are connected in the constraints, namely via the constraint that $Y_{ij} + Y_{ji} = 1$. These arcs are connected, because they form an arc pair, being the complementary arc of each other. The fortunate finding is that arcs from different arc pairs are **not** connected anywhere in the constraints of $LR(\lambda)$. We conclude that (1) the feasible region and (2) the non-constant terms of the objective function are separable for each arc pair.

Therefore, we find the optimal solution to $LR(\lambda)$ by sequentially solving a sub-problem related to each arc pair. The sub-problem related to arc pair $(i,j),(j,i) \in A$ is denoted by $SP_{ij}(\lambda)$:

$$\begin{aligned} SP_{ij}(\lambda) \;\; = \min \quad & \Big\{ \sum_{p \in P} \Big((\lambda_i^p + c_{ij}) X_{ij}^p \;-\; \lambda_i^p X_{ji}^p \Big) \;\; + \\ & \sum_{p \in P} \Big((\lambda_j^p + c_{ji}) X_{ji}^p \;-\; \lambda_j^p X_{ij}^p \Big) \quad : \\ & Y_{ij} + Y_{ji} = 1 \;;\; Y_{ij}, Y_{ji} \in \{0,1\} \;;\; X_{ij}^p, X_{ji}^p \in \{0,1\}\,,\, \forall p \in P \;; \\ & X_{ij}^p \le Y_{ij}\,,\, \forall p \in P \;;\; X_{ji}^p \le Y_{ji}\,,\, \forall p \in P \Big\}. \end{aligned}$$

The sub-problem can be written slightly more compact. The final form of the sub-problem is exhibited in Equation (4) below:

$$\begin{aligned} SP_{ij}(\lambda) \;=\; \min \quad & \Bigg\{ \sum_{p\in P} \Big(\big(\lambda_i^p + c_{ij} - \lambda_j^p\big) X_{ij}^p \;+\; \big(\lambda_j^p + c_{ji} - \lambda_i^p\big) X_{ji}^p \Big) \quad : \\ & Y_{ij} + Y_{ji} = 1 \;\;;\;\; Y_{ij}, Y_{ji} \in \{0,1\} \;\;;\;\; X_{ij}^p, X_{ji}^p \in \{0,1\}\,, \forall p \in P \;; \\ & X_{ij}^p \leq Y_{ij}\,, \forall p \in P \;\;;\;\; X_{ji}^p \leq Y_{ji}\,, \forall p \in P \Bigg\}\,. \end{aligned} \tag{4}$$

Note that arcs (i, j) and (j, i) have both been incorporated in $SP_{ij}(\lambda)$ and hence a single sub-problem is solved for each arc pair. The way to solve a sub-problem $SP_{ij}(\lambda)$ is illustrated now. We initialize $\zeta(i, j)$ and $\zeta(j, i)$ equal to 0. The aim is to find out whether we should choose $Y_{ij} = 1$ or $Y_{ji} = 1$, and to discover the corresponding effect on the value of the objective function. The terms belonging to an arbitrary OD pair $p \in P$ in Equation (4) are:

$$\big(\lambda_i^p + c_{ij} - \lambda_j^p\big) X_{ij}^p + \big(\lambda_j^p + c_{ji} - \lambda_i^p\big) X_{ji}^p \quad .$$

We have to take the specified feasible region of SP_{ij} into account. If we would be in a position wherein arc (i, j) is enabled ($Y_{ij} = 1$):

- We opt for $X_{ij}^p = 1$, if $\lambda_i^p + c_{ij} - \lambda_j^p < 0$
- We opt for $X_{ij}^p = 0$, if $\lambda_i^p + c_{ij} - \lambda_j^p \geq 0$

Therefore, we add $min\{\lambda_i^p + c_{ij} - \lambda_j^p\,, 0\}$ to $\zeta(i, j)$. Similarly, we add $min\{\lambda_j^p + c_{ji} - \lambda_i^p\,, 0\}$ to $\zeta(j, i)$. After repeating this exact same procedure for all OD pairs $p \in P$, we choose the arc that has a lower value of $\zeta(\cdot)$, as (overall) this arc can cause the largest decrease in the objective value corresponding to the sub-problem. The optimal value for the sub-problem is denoted by $SP_{ij}^*(\lambda)$ and is expressed in Equation (5):

$$SP_{ij}^*(\lambda) = \min\Bigg\{ \sum_{p\in P} \min\{\lambda_i^p + c_{ij} - \lambda_j^p\,, 0\}\;,\; \sum_{p\in P} \min\{\lambda_j^p + c_{ji} - \lambda_i^p\,, 0\} \Bigg\}\,. \tag{5}$$

We have currently demonstrated that the Lagrangian relaxation that results after relaxing these particular constraints in this particular MIP formulation leads to a problem that can be split up into easily solvable sub-problems. These sub-problems are constantly solved, given the fixed Lagrange multipliers λ_i^p. The general purpose of the Lagrangian method is to adjust the Lagrange multipliers in such a way that the best possible lower bound (from now on denoted by LR) is eventually found. The formal definition of LR is given by

$$LR \;=\; \max_{\lambda} \Big\{ LR(\lambda) \Big\} \;, \tag{6}$$

which shows that the goal is to find the Lagrange multipliers for which $LR(\lambda)$ is maximized. We have used the subgradient method (as described in Fisher (1981)) to update the Lagrange multipliers throughout the iterations.

Comparing the Lagrangian relaxation to the LP relaxation

Although we have successfully established an efficient manner to tackle this particular Lagrangian relaxation, the theoretical quality of the lower bound provides a less optimistic perspective. In the remainder of this subsection, we prove that the lower bound LR resulting from the Lagrangian method is not better than the lower bound supplied by the LP relaxation of the original MIP formulation. More specifically, it is proven that these two lower bounds are equal. This statement is shown in Theorem 4.

Theorem 4. *For the EOP-ODP, LR is equal to the lower bound provided by the LP relaxation.*

Proof. We utilize a well-established theorem (e.g. mentioned in Fisher (1981)). This theorem states that LR is equal to the lower bound provided by the LP relaxation, if the value of $LR(\lambda)$ is not reduced when its integrality conditions for the variables are dropped.
In order to prove Theorem 4, we will thus prove that the value of $LR(\lambda)$ is not affected when the integrality conditions of the variables are dropped, by considering any arbitrary sub-problem belonging to arc pair $(i,j),(j,i) \in A$. Under integrality conditions for both the Y_{ij} and X_{ij}^p variables, the optimal value for a sub-problem was denoted by $SP_{ij}^*(\lambda)$ and given by the above Equation (5). Say that one chooses any value $\delta \in [0,1]$ for Y_{ij}, causing Y_{ji} to attain the value $1-\delta$. Recall that the sub-problem in the case where all variables are integral was analyzed in detail above, resulting in the expression for $SP_{ij}^*(\lambda)$. Fortunately, the sub-problem wherein $Y_{ij}=\delta$ and $Y_{ji}=1-\delta$ can analytically be solved in a similar way. One can derive that the following expression constitutes the optimal value for this sub-problem:

$$\begin{aligned}
& \sum_{p\in P} \delta \cdot \min\{\lambda_i^p + c_{ij} - \lambda_j^p\,,\,0\} \;+\; \sum_{p\in P} (1-\delta)\cdot \min\{\lambda_j^p + c_{ji} - \lambda_i^p\,,\,0\} \\
= \;& \delta \cdot \sum_{p\in P} \min\{\lambda_i^p + c_{ij} - \lambda_j^p\,,\,0\} \;+\; (1-\delta)\cdot \sum_{p\in P} \min\{\lambda_j^p + c_{ji} - \lambda_i^p\,,\,0\} \\
\geq \;& \delta \cdot SP_{ij}^*(\lambda) \,+\, (1-\delta)\cdot SP_{ij}^*(\lambda) \\
= \;& SP_{ij}^*(\lambda)\,.
\end{aligned}$$

This proves that relaxing the integrality constraints on the variables does not lead to a smaller value for any arbitrary sub-problem, and hence will not lead to a smaller value for $LR(\lambda)$. Therefore, the proof for Theorem 4 is complete. □

We conclude that the final lower bound LR provided by this Lagrangian method is equal to the lower bound provided by the LP relaxation of our MIP formulation.

6 Data

To the best of our knowledge, consistent with the fact that the EOP-ODP is not a widely studied problem yet, there are currently no public datasets available. Therefore, it is necessary to generate our own instances. In practice, it is more likely for a connection between two nodes to exist when the distance between the corresponding nodes is small. To translate this aspect into the instances, every node $i \in V$ is assigned an (x_i, y_i) pair of coordinates. Here, x_i and y_i are independently generated from the Uniform(0,1) distribution. The Euclidean distance between nodes is utilized.

The following procedure is used to determine which nodes will be connected by edges. For each node $i \in V$, a normally distributed variable with mean μ and standard deviation σ is drawn. This normally distributed variable is rounded to the nearest integer to constitute the degree of the node. All integers smaller than 2 will be replaced by 2, to assure that each node has at least two connections. The number of edges belonging to node i is denoted by ϵ_i. The creation of ϵ_i edges advances as follows. Constantly, a random other node j is drawn. Let γ denote a positive-valued parameter and $d(i, j)$ denote the distance between nodes i and j. The probability of the edge between nodes i and j being accepted is $e^{-\gamma \cdot d(v_1, v_2)}$. The condition of two nodes sharing an edge being more likely when the distance is smaller is clearly imposed in this formula. The procedure of drawing random nodes j is continued until the desired amount of edges (ϵ_i) for node i is fulfilled. The procedure for the arbitrary node i is repeated for all nodes. Note that this approach renders it possible for certain nodes to obtain a higher degree than initially determined. Analyzing this entire procedure, one can conclude that a higher value of γ reduces the expected average distance of the edges between nodes.

Finally, the task of creating the OD pairs remains. The amount of OD pairs that is desired is supplied beforehand. Repeatedly, a pair of vertices i and j $(i, j \in V, i \neq j)$ is drawn randomly. The OD pair with origin node i and destination node j is accepted according to a certain probability $\omega(i, j)$. This procedure is executed until the desired amount of OD pairs is reached. We have chosen the following characteristics of instances to vary:

1. The amount of nodes:
    - Option A: 50 nodes
    - Option B: 150 nodes
    - Option C: 250 nodes
2. The parameter μ:
    - Option A: $\mu = 2$
    - Option B: $\mu = 2.5$
    - Option C: $\mu = 3$

3. The parameter σ:
    - Option A: $\sigma = 0.5$
    - Option B: $\sigma = 1$
    - Option C: $\sigma = 2$
4. The parameter γ:
    - Option A: $\gamma = 5$
    - Option B: $\gamma = 7.5$
    - Option C: $\gamma = 10$
5. The amount of OD pairs:
    - Option A: $max\{n, \frac{n(n-1)}{100}\}$ OD pairs
    - Option B: $max\{n, \frac{n(n-1)}{30}\}$ OD pairs
    - Option C: $max\{n, \frac{n(n-1)}{10}\}$ OD pairs
6. The relationship between OD pairs and distance:
    - Option A: Mainly accept OD pairs between nodes with a small distance. This is done by setting $\omega(i,j) = e^{-d(i,j)}$.
    - Option B: Mainly accept OD pairs between nodes with a large distance. This is done by setting $\omega(i,j) = e^{d(i,j)-1}$.
    - Option C: Aim for no relationship between OD pairs and distance, achieved by setting $\omega(i,j) = 1$.

We have to note a slight adaptation that we incorporate in the process of generating the instances. As the instances are created to compare the performance of several methods, it is not useful to work with instances for which no feasible solution exists. After generation, Algorithm 1 (which was introduced in Section 4) is utilized to examine whether an instance is infeasible. If this is the case, we simply generate an entirely new instance, until we arrive at an instance for which a feasible solution exists. The instances are named according to the options that they represent. For example, the instance $ABCABC$ signifies 50 nodes, $\mu = 2.5$, $\sigma = 2$ etc. Among all possibilities, we selected a balanced set of instances, by trial and error. The descriptive statistics of our dataset of instances are exhibited in Table 1. The instances with a higher amount of nodes, edges and OD pairs are found in the lower part of Table 1.

Table 1: Descriptive statistics of the instances.

Instances	Nodes	Edges	OD pairs
ABBAAA	50	96	50
AAAABC	50	70	81
AACBBB	50	84	81
ACBBBA	50	99	81
BCCAAC	150	323	223
AACACB	50	91	245
CBBBAB	250	448	622
CBCAAA	250	494	622
BAACBA	150	216	745
BBCABA	150	287	745
CAACBC	250	349	2075
CABABB	250	381	2075
CCBBBA	250	508	2075
CCCBBB	250	571	2075
BBCCCB	150	302	2235
BCCCCC	150	329	2235
BCCBCC	150	342	2235
CBCCCB	250	498	6225
CCBCCB	250	505	6225
CCCACA	250	557	6225

7 Results

This section is devoted to exhibiting results on the performance of the methods described in Section 5. The results were run on a HP Pavilion computer with 16.0 GB RAM and an AMD Ryzen 5 processor. The operating system is Windows 10. Java is used as programming language. The MIP formulation is solved by version 12.6.3 of the commercial solver CPLEX.

7.1 Variants

To obtain information on the performance of the methods under different time restrictions, two variants of each method are implemented. Let *MIP(1h)* denote the MIP formulation with a time limit of one hour and *MIP(3h)* represent the MIP formulation with a time limit of three hours. Similarly, *SH(1h)* and *SH(3h)* respectively correspond to the allocation of one hour and three hours to the Simple Heuristic. Furthermore, *ACO(1h)* and *ACO(3h)* respectively denote the Ant Colony Optimization algorithm with a time limit of one hour and three hours. Last, in Section 4, we invented a way that guarantees the construction of a feasible solution to the EOP-ODP, if a feasible solution exists. This method is labeled as *FEAS*.

7.2 Parameters for the ACO algorithm

Next to the time limit parameters, which are incorporated in the variants of each method, the ACO algorithm contains numerous additional parameters. Unless stated otherwise, we used trial-and-error to assign values to these additional parameters. The amount of paths generated by the Vanhove heuristic in Step 0 (denoted by k) is initialized as 20. The number of ants N is chosen as 50. Furthermore, the parameter α is set to 1 and β is set to 2. The remaining parameters correspond to the pheromone level update of the paths. The parameter π is set to 10, meaning that the global best solution is used for the pheromone update once in every ten iterations. As mentioned in Subsection 5.3.4, higher values of ρ lead to a better exploration of the solution space and a larger amount of time before reaching high-quality solutions. Therefore, we equip the variants *ACO(1h)* and *ACO(3h)* with separate values of ρ. The value of $\rho = 0.96$ is used in *ACO(1h)*, whereas $\rho = 0.98$ is used in *ACO(3h)*. For τ_{max}, the maximum pheromone level of a path, we follow the approach of Stützle & Hoos (2000), who have elected a value according to the asymptotically maximum pheromone level:

$$\tau_{max} = \frac{1}{1-\rho}\,\frac{1}{f(s^{gb})} \quad ,$$

where $f(s^{gb})$ denotes the solution value of the currently best discovered solution. This implies that the value of τ_{max} is altered every time an improving solution is found. The final aspect that has to be discussed is the initialization of the pheromone level of the paths, represented by the parameter τ^0. Stützle & Hoos (2000) have conducted various experiments, which indicate that initializing τ^0 equal to τ_{max} is a suitable option. In this way, the pheromone level difference between the paths remains relatively small in the beginning of the algorithm, which encourages a broad exploration of the solution space. Practically, as no value for τ_{max} is present at the start of the algorithm, we set τ^0 to a large value for all paths, and then the pheromone level will be reduced to τ_{max} after the first solution has been found.

7.3 Performance of the variants

In this part, the performance of the several variants is presented and discussed. For variant *ACO(1h)*, the best discovered solution value $f(s^{gb})$ is reported. The performance of the other variants is reported relative to the $f(s^{gb})$ of *ACO(1h)*. Let $f(s^{ot})$ denote the best solution value by an other variant. Then, f_{dif} is the percentage change of the best discovered objective value belonging to the other variant, compared to $f(s^{gb})$. This is calculated by the following formula:

$$f_{dif} = 100 \times \frac{f(s^{ot}) - f(s^{gb})}{f(s^{gb})} \quad .$$

Note that a positive value of f_{dif} coincides with the other variant providing a worse solution than *ACO(1h)*, whereas a negative value of f_{dif} corresponds to the other variant providing a better solution. The results are reported in Table 2 below. The result of the best performing variant for an instance is highlighted in bold. A value of "–" is used to indicate the scenario wherein a certain variant was not able to provide any solution within its time limit. When we have knowledge that a certain variant has provided the optimal solution, this is indicated by a "$*$" symbol.

Table 2: Results of the variants.

	ACO(1h)	**ACO(3h)**	**MIP(1h)**	**MIP(3h)**	**SH(1h)**	**SH(3h)**	**FEAS**
Instances	$f(s^{gb})$	f_{dif}	f_{dif}	f_{dif}	f_{dif}	f_{dif}	f_{dif}
ABBAAA	44.97	0.00	**-0.04***	**-0.04***	0.00	0.16	234.16
AAAABC	130.56	0.00	**-0.02***	**-0.02***	0.13	**-0.02***	113.19
AACBBB	**83.92***	**0.00***	**0.00***	**0.00***	0.80	0.20	147.35
ACBBBA	62.06	-0.52	**-0.56***	**-0.56***	-0.08	-0.08	190.70
BCCAAC	**249.67**	0.04	4.65	0.32	4.92	4.48	459.39
AACACB	320.92	-0.35	**-0.42***	**-0.42***	1.52	1.69	157.23
CBBBAB	835.34	**-0.24**	–	85.52	5.74	6.60	612.42
CBCAAA	840.67	**-1.05**	–	11.87	6.31	5.50	790.51
BAACBA	1063.87	**-0.17**	–	4.34	4.09	4.52	220.40
BBCABA	967.97	**-0.59**	–	4.98	6.38	6.28	445.98
CAACBC	3763.64	**-1.21**	–	–	4.22	4.85	250.94
CABABB	4439.76	**-1.06**	–	–	6.44	5.56	418.85
CCBBBA	2341.66	**-0.85**	–	–	7.20	7.60	627.25
CCCBBB	**2245.39**	0.02	–	100.81	6.08	5.78	694.18
BBCCCB	2519.38	**-0.33**	–	–	4.15	4.21	335.43
BCCCCC	**2097.97**	0.51	–	–	5.62	4.88	329.58
BCCBCC	2189.04	**-0.08**	–	–	5.66	4.52	460.36
CBCCCB	6695.22	**-1.27**	–	–	5.24	4.50	534.06
CCBCCB	7156.24	**-0.83**	–	–	4.90	4.00	548.37
CCCACA	7739.55	**-0.07**	–	–	6.29	6.48	808.03
Average	N.A.	**-0.40**	N.A.	N.A.	**4.28**	**4.09**	**418.92**

Recall that the instances with a higher amount of nodes, edges and OD pairs are found in the lower part of Table 2. Furthermore, we remind the reader that the performance of all variants is denoted relative to $f(s^{gb})$, which is the solution value found by the *ACO(1h)* variant.

Comparing the exact method to the heuristic methods

The results of Table 2 reveal that, as expected, the MIP formulation has performed well on the small instances and poorly on the larger instances. The MIP formulation is an exact method, meaning that it takes steps (by means of a branch-and-bound tree) towards finding the optimal solution and is able to provide definite proof that an optimal solution has been found. Conversely, the Simple Heuristic and ACO algorithm are heuristic methods. This means that, instead of operating in a structured manner designed to eventually prove that the optimal solution has been found, these two methods simply aim to constantly find improving solutions. For a total of five small instances, the exact nature of the MIP formulation proved itself to be advantageous. Whereas the Simple Heuristic and ACO variants could overlook certain solutions due to their heuristic nature, the MIP formulation explored all possibilities and managed to deliver the optimal (or near-optimal) solution within the time limit. We put the advantage of the MIP formulation on the small instances in perspective, by noting that there are only slight differences between the solution values of the MIP variants and the Simple Heuristic and Ant Colony Optimization variants (e.g. -0.04% instead of 0.00% on the upper instance in Table 2). Contrary to the results on the small instances, the exact MIP formulation appeared to be highly disadvantageous for the larger instances. Due to the substantially large amount of variables and constraints present in the MIP formulation of these instances, the branch-and-bound tree could generally not be explored sufficiently profound within the time limit, causing the obtained results to be very poor. For the variant *MIP(1h)*, no feasible solution was retrieved for 14 out of the 20 instances. Even with the three hours of computation time allocated to *MIP(3h)*, no feasible solution could be retrieved for nine instances. Additionally, for three of the remaining instances, the deterioration of the solution value retrieved by *MIP(3h)* was at least 10%, when compared to $f(s^{gb})$. Deteriorations of at least 10% never occurred for both of the Simple Heuristic variants and the Ant Colony Optimization variant with a time limit of three hours. Overall, we can thus conclude that the variants of the MIP formulation were vastly outperformed by the Simple Heuristic and ACO variants on our generated set of instances.

Comparing the two heuristic methods

We have established that it is worthwhile to use a heuristic approach instead of the exact MIP approach for our generated set of instances. In Section 5.2, we defined the Simple Heuristic as a benchmark for the more sophisticated Ant Colony Optimization algorithm. We thus expected the ACO algorithm to perform better than the Simple Heuristic. The results indicate that this expectation has clearly been fulfilled. The *ACO(1h)* has found a better or equal solution than *SH(1h)* for 19 out of 20 instances. The average deterioration of the solution quality of *SH(1h)* was 4.28% compared to $f(s^{gb})$. When comparing the *ACO(3h)* and *SH(3h)* variants, we observe that the Ant Colony Optimization variant has found a better or equal solution for 19 out of 20 instances. When comparing to $f(s^{gb})$, the average solution value of *ACO(3h)* is 0.40% lower, whereas the variant of *SH(3h)* is associated with an average solution value that is 4.09% higher. We can conclude that the Ant Colony Optimization algorithm clearly outperforms the Simple Heuristic.

Comparing the variants within both heuristic methods

For both the SH and the ACO heuristics, we observe that the variant with a larger time limit occasionally retrieves worse solutions. For example, on instance *CCCACA*, the *SH(1h)* resulted in a 6.29% increase with respect to $f(s^{gb})$, whereas the *SH(3h)* resulted in a 6.48% increase. An example related to the ACO variants is instance *BCCCCC*, where the variant *ACO(3h)* provided a solution value that constituted a 0.51% increase with respect to the *ACO(1h)* variant. These findings are a result of the stochastic nature of both heuristics. This stochastic nature is present in the order in which to consider the OD pairs (in both SH and ACO) and the path that is selected for a certain OD pair (in ACO). The fact that the variants with smaller time limits occasionally find better solutions should thus not be conceived as contradictory. Overall, the *SH(3h)* variant found a solution that was at least as good as the solution of the *SH(1h)* variant for 12 of the 20 instances. Furthermore, compared to $f(s^{gb})$, the average increase in solution value was 4.28% for the *SH(1h)* variant and 4.09% for the *SH(3h)* variant. We can thus conclude that the Simple Heuristic variant with three hours of allocated time limit has only slightly outperformed the Simple Heuristic variant with one hour of allocated time limit. Comparing both ACO variants, the *ACO(3h)* variant found a solution that was at least as good for 17 of the 20 instances. The average solution value difference of *ACO(3h)*, with respect to the *ACO(1h)*, was -0.40%. This leads us to conclude that the ACO variant with three hours of allocated time has moderately outperformed the ACO variant with one hour of allocated time.

In advance, one could have expected the two ACO variants to be further distinguished in performance than the two SH variants. Recall that, in the ACO algorithm, we are able to impose a trade-off between computation time and solution quality via the parameter ρ. A higher value of ρ is typically associated with a larger concentration on solution quality. Logically, we opted for a higher value of ρ for the *ACO(3h)* variant. Then, whereas the iterations of the Ant Colony Optimization algorithm jointly work towards promising areas of the solution space (by using ρ to update the pheromone level of paths), the iterations of the Simple Heuristic operate entirely independently. This implies that the allocation of three hours (instead of one hour) of computation time to the Ant Colony Optimization algorithm results in a distinguishable framework that is more and more focused on solution quality, while allocating three hours (instead of one hour) of computation time to the Simple Heuristic merely results in the execution of approximately three times as many iterations.

Assessing the performance of the *FEAS* method

Considering all different methods that were introduced in this thesis, the *FEAS* method possesses a special property. Given that a feasible solution to an instance of the EOP-ODP exists, it is the only method that can provide certainty in delivering a feasible solution in polynomial time. The fact that the *FEAS* method exclusively focuses on feasibility is expected to be at the cost of the solution quality. Table 2 clearly confirms this deterioration in solution quality. The results indicate that the *FEAS* method is unable to compete with the other methods. The *FEAS* method produced solution values that formed an increase of 418.92% on average, compared to the *ACO(1h)* variant. Additionally, as both variants of the Simple Heuristic and the Ant Colony Optimization delivered a feasible solution for all instances, the *FEAS* method was unable to positively distinguish itself in this area. We conclude that the *FEAS* method should be perceived useful in detecting whether a feasible solution to an instance of the EOP-ODP exists, but is highly unlikely to be a competitive method for providing solutions to an instance.

7.4 Lower bounds

Throughout this section, we have established that the *ACO(3h)* variant is our most promising method for the EOP-ODP. In this part, we briefly assess the extent of improvement that can be made on the solutions of the *ACO(3h)* variant. The solution values are compared to the best lower bound that we found. These lower bounds result from running the MIP formulation with a time limit of three hours (i.e. the *MIP(3h)* variant). The best discovered lower bound is denoted by LB. The objective value corresponding to the solution found by the *ACO(3h)* variant is denoted by $f(ACO^{3h})$. The relative increase is calculated as follows:

$$f_{inc} = 100 \times \frac{f(ACO^{3h}) - LB}{LB} \quad .$$

The value of f_{inc} should be regarded as an upper bound for the gap between the solution value found by the *ACO(3h)* variant and the optimal solution value. Moreover, logically, f_{inc} is restricted to non-negative values. The results are shown in Table 3.

For two of the 20 instances, we needed to use our Lagrangian relaxation of Subsection 5.4 to extract a lower bound. These two instances were substantially large, such that the root node of the MIP formulation could not be solved within three hours of computation time. Our Lagrangian relaxation turned out to be a viable alternative in these cases, as it was able to converge fast, and thus provide a lower bound within a small amount of time. As the proof in Subsection 5.4 illustrates, the lower bound found by the Lagrangian relaxation is equal to the lower bound that would have been obtained by solving the LP relaxation of the MIP formulation.

Table 3: The lower bounds, compared to the *ACO(3h)* variant

Instances	LB	f_{inc}
ABBAAA	44.95	0.04
AAAABC	130.53	0.02
AACBBB	83.92	0.00
ACBBBA	61.71	0.05
BCCAAC	240.42	3.90
AACACB	319.56	0.07
CBBBAB	718.70	15.96
CBCAAA	731.02	13.80
BAACBA	892.77	18.97
BBCABA	829.75	15.97
CAACBC	2966.91	25.32
CABABB	3533.26	24.32
CCBBBA	1982.37	17.12
CCCBBB	1989.51	12.88
BBCCCB	2196.98	14.30
BCCCCC	1900.08	10.98
BCCBCC	1968.15	11.14
CBCCCB	5760.14	14.76
CCBCCB	6160.88	15.19
CCCACA	6387.44	21.08

For the smaller instances in Table 3, the solution value of the best solution found by the *ACO(3h)* variant was particularly close to the lower bound. A total of five instances were solved to optimality by the MIP formulation. These are the only instances for which f_{inc} is certain to correspond with the gap between the solution value found by the *ACO(3h)* variant and the optimal solution value. All of the five instances are associated with values of f_{inc} that are smaller than 0.10%. For the bottom 14 instances, on the other hand, the value of f_{inc} was always at least 10%, with a maximum of 25.32%. On these instances, we can not rule out that substantial improvement is possible.

8 Conclusions

In this thesis, the Edge Orientation Problem with Origin-Destination Pairs (EOP-ODP) was investigated. The EOP-ODP entails the assignment of an orientation to the edges of an initially undirected graph, such that directed paths exist for all Origin-Destination (OD) pairs and the total distance of these OD paths is minimized. The first part of the thesis was devoted to the derivation of two theoretical results. The complexity of the EOP-ODP was established to be strongly $\mathcal{NP}$-hard. This result validates the search for heuristic solution approaches. Next, we developed an algorithm that analyzes whether a feasible solution to an instance of the EOP-ODP exists. Our algorithm can provide certainty on whether a feasible solution exists, in polynomial time. We extended this algorithm by describing a polynomial-time procedure that can be executed to actually create a feasible solution, given that one exists.

In the second part of the thesis, an MIP formulation, a simple heuristic and an Ant Colony Optimization algorithm were designed. To the best of our knowledge, there were no existing datasets for the EOP-ODP, causing the need for us to develop a dataset ourselves. During the process of developing a dataset, it is convenient to utilize the previously reported algorithm on whether a feasible solution exists, when one wants to avoid the inclusion of infeasible instances in the dataset. The 20 instances of our dataset were used to test the performance of the three solution approaches. Each solution approach was tested twice, once with a time limit of one hour and once with a time limit of three hours. The exact MIP formulation proved to be slightly advantageous for small instances, but was heavily outperformed by the two other (heuristic) solution approaches on larger instances. Overall, the Ant Colony Optimization algorithm clearly established itself as the most successful solution approach for the EOP-ODP.

The instances in our dataset did not demand a graph with highly specific properties. However, certain real-life scenarios related to the EOP-ODP are associated with a specific type of graph. For example, the scenario of assigning an orientation to streets in a city centre can be modelled by a *planar graph* (i.e. a graph that can be drawn in such a way that its edges only intersect at endpoints). Exploring the performance of each of our methods on a dataset of planar graphs can thus be considered a meaningful topic for future research.

Another interesting direction for future research is the generation of lower bounds for the EOP-

ODP. In this thesis, we have concluded that our Ant Colony Optimization algorithm has outperformed our MIP formulation and our simple heuristic. However, for the larger instances in our dataset, the gaps between the solution values of the Ant Colony Optimization algorithm and the best discovered lower bounds were substantial. This implies that we can not rule out that considerable improvement is possible on these instances. Generating tight lower bounds can provide more information on the proximity of the solution values of the Ant Colony Optimization algorithm to the optimal solution values. We have provided a contribution to the knowledge on lower bounds for the EOP-ODP. For our MIP formulation, a Lagrangian relaxation was explored. We proved that this particular Lagrangian relaxation always leads to the same lower bound as the LP relaxation of the MIP formulation. The Lagrangian relaxation did turn out to be a valuable addition to the results of this thesis. Generally, the best discovered lower bounds were obtained by running the MIP formulation with a time limit of three hours. However, for two of the 20 instances, the root node of the MIP formulation could not be solved within three hours, and therefore we needed our Lagrangian relaxation to obtain a lower bound. The Lagrangian relaxation delivered the lower bound in a small amount of time. In conclusion, in this thesis we have identified our Lagrangian relaxation as useful to some large instances, but we have also proven that its lower bound is equal to the lower bound of the LP relaxation. We consider the development of methods that lead to tighter lower bounds for the EOP-ODP as a relevant direction for future research.

References

Atallah, M. J. (1984). Parallel strong orientation of an undirected graph. *Information Processing Letters*, *18*(1), 37-39.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, *2*(4), 353–373.

Chan, T.-H. H., Sozio, M., & Sun, B. (2021). Distributed approximate k-core decomposition and min–max edge orientation: Breaking the diameter barrier. *Journal of Parallel and Distributed Computing*, *147*, 87–99.

Chondrogiannis, T., Bouros, P., Gamper, J., & Leser, U. (2015). Alternative routing: k-shortest paths with limited overlap. In *Proceedings of the 23rd sigspatial international conference on advances in geographic information systems* (pp. 1–4).

Chvátal, V., & Thomassen, C. (1978). Distances in orientations of graphs. *Journal of Combinatorial Theory, Series B*, *24*(1), 61-75.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management science*, *27*(1), 1–18.

Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. , *34*(3), 596–615.

Ito, T., Miyamoto, Y., Ono, H., Tamaki, H., & Uehara, R. (2013). Route-enabling graph orientation problems. *Algorithmica*, *65*, 317-338.

Kwok, P. K., Liu, Q., & West, D. B. (2010). Oriented diameter of graphs with diameter 3. *Journal of Combinatorial Theory, Series B*, *100*(3), 265-274.

Mitchell, S. L., Cockayne, E. J., & Hedetniemi, S. T. (1979). Linear algorithms on recursive representations of trees. *Journal of Computer and System Sciences*, *18*(1), 76–85.

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.

Stützle, T., & Hoos, H. H. (2000). Max–min ant system. *Future generation computer systems*, *16*(8), 889–914.

Surmacs, M. (2017). Improved bound on the oriented diameter of graphs with given minimum degree. *European Journal of Combinatorics*, *59*, 187-191.

Taccari, L. (2016). Integer programming formulations for the elementary shortest path problem. *European Journal of Operational Research*, *252*(1), 122-130.

Vanhove, S., & Fack, V. (2012). An effective heuristic for computing many shortest path alternatives in road networks. *International Journal of Geographical Information Science*, *26*(6), 1031–1050.

Voss, C., Moll, M., & Kavraki, L. (2015). A heuristic approach to finding diverse short paths. *Proceedings - IEEE International Conference on Robotics and Automation*, *2015*, 4173-4179.

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Science*, *17*(11), 712–716.

Yen, W. C.-K. (2006). The edge-orientation problem and some of its variants on weighted graphs. *Information Sciences*, *176*(19), 2791-2816.

Yen, W. C.-K. (2007). Edge-Orienting on Split, Planar and Treelike Graphs. *The Computer Journal*, *50*(3), 357-368.