BACHELOR

Routing Autonomous Agricultural Vehicles and 1-tree Lagrangian Relaxation

Bakens, Dean

Award date:
2020

Routing Autonomous Agricultural Vehicles and 1-tree Lagrangian Relaxation

Author:
Dean Bakens
1026726
d.bakens@student.tue.nl

Supervisor:
C.A.J. Hurkens
c.a.j.hurkens@tue.nl

December 5, 2020

Abstract

In this thesis, one aims to create routing software for an autonomous agricultural vehicle on an agricultural field. This autonomous agricultural vehicle has to perform work on a set of required edges on an agricultural field and in order to model this, this problem will be translated to the Rural Postman Problem. Firstly, this problem will be tackled using a heuristic algorithm based on Monte Carlo methods and some slight alterations to this. Secondly, one also wants to assess the quality of the solutions obtained using this algorithm in the form of a lower bound. This lower bound will at first simply be obtained using a minimum spanning tree. Later, this bound will be tightened using a Lagrangian relaxation that uses an elaboration of the minimum spanning tree. Lastly, an ad hoc approach will be described to solve the routing problem at hand that combines the altered heuristic algorithm with the Lagrangian relaxation in order to improve results.

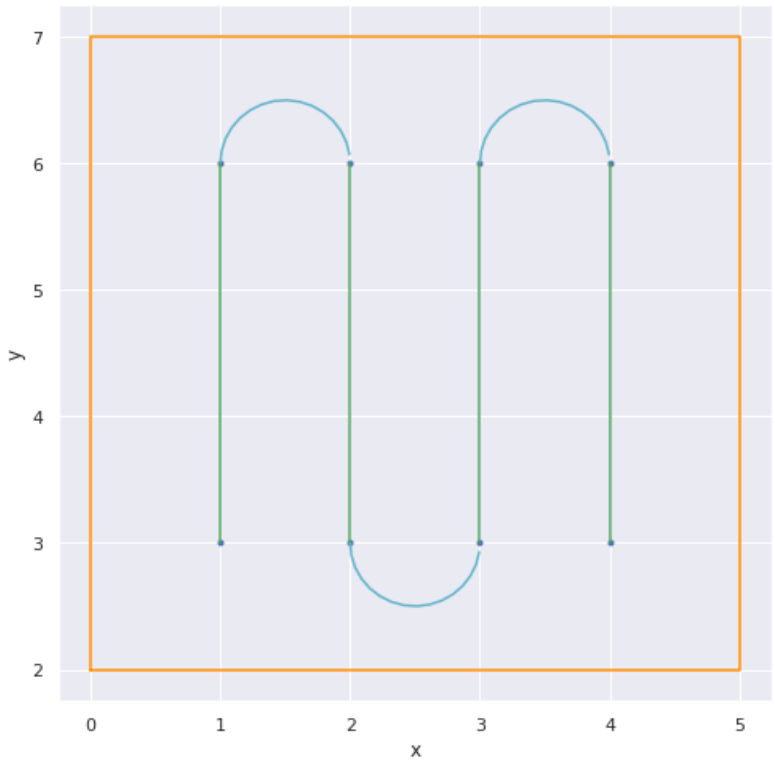
Contents

1 Introduction

The company Agxeed [2] is designing an autonomous agricultural vehicle. In order to design such an autonomous agricultural vehicle, one has to create the routing software that calculates the path that the vehicle has to travel on some agricultural field. This software is designed by the company Phact [3].

In general, these agricultural fields are divided into two parts, namely the main land and the head land. The main land is the area of the field where most crops are grown and the head land is the land surrounding the main land, which at first is used for turning the vehicle, and eventually for growing more crops as well.

In this setting, an agricultural field is the inside area of a field defined within a certain boundary. This boundary is a polygon, described as a path along the points $P_0, P_1, \cdots, P_n = P_0$, in which the area is bounded by the straight line segments (P_i, P_{i+1}), for $i = 0, \cdots, n-1$. Now, the main land is an area within the agricultural field which is located at a fixed distance from the boundary, which can also be described as a polygon. Within the main land, there are a certain number of parallel line segments, also a fixed distance apart from one another, that the autonomous agricultural vehicle has to perform its work on. Moreover, in the head land, there are also a certain number of equidistant line segments, however these are parallel to the boundary of the field. Now, the goal is to, given a starting point on the boundary of the agricultural field, find a route of minimal length in which the vehicle, from this starting point, visits every line segment in the main land exactly once, at which the direction is irrelevant. After visiting the edges in the main land, the vehicle visits the line segments in the head land inside out, after which the vehicle will thus be located on the polygon, which is the boundary of the agricultural field again. A very simple example of a route performed on the main land of some arbitrarily generated field can be seen in figure 1a. Furthermore, a real-world example of such an agricultural field with some route performed on it can be seen in figure 1b, in which the blue lines represent the turns connecting the required edges.



(a) Simple example route performed on the main land of some arbitrary field

(b) Example of a complete route performed on an agricultural field

Figure 1: An illustrative and a real-world example of routes performed on some fields

Essentially, the focus of this thesis will be placed on minimizing the distance travelled on the main

land, which is done by minimizing the length of the turns made to connect the required edges in the main land. This will be the focus as this part of the route has the most room for improvement. This might seem rather simple, as one might opt for a naive solution, which would be defined as a solution in which every required edge in the main land is connected to the required edge next to it. However, in practice this is far from optimal as this would require the vehicle to make turns that would almost represent a closed circle, where a turn that is slightly larger than half a circle would already suffice. This would essentially mean that the vehicle has to make unnecessarily big turns due to the turning radius of the vehicle.

Furthermore, in order to minimize the distance travelled by the vehicle, one has to define and actually calculate the turns the vehicle can make, as this is essentially what has to be minimized. This will be done using Dubins curves [13]. These Dubins curves are calculated between every pair of nodes in the main land and these can be considered to be edges as well.

In this thesis, the agricultural fields will be represented as undirected graphs, in which the nodes in the main land connected by the required edges will represent the set of nodes and the edges between these nodes combined with the turns described above are in the edge set. Doing this enables one to use all sorts of graph related problems and algorithms to perform on these graphs in order to obtain results regarding its routing. Moreover, the given problem can now be described as the Rural Postman Problem, which is a practical extension of the Chinese Postman Problem, attempting to find a shortest route over a set of required edges, which in this case are the edges on which the agricultural vehicle has to perform its labour. The Rural Postman Problem is part of the set of NP-complete problems, meaning that there is no known way to find a solution to the problem efficiently. Moreover, if one assumes that $P \neq NP$, as is widely accepted, it cannot admit a polynomial time algorithm to solve the problem [14].

Now, using this representation, a heuristic algorithm based on Monte Carlo methods for the Rural Postman Problem has already been performed in order to improve results [8], which will be discussed later, alongside some improvements that can be made.

Lastly, this representation of the problem into graphs also enables one to use the notion of a Minimum Spanning Tree (MST), which, using some alterations and extensions, enables one to compute lower bounds for the problem and in certain cases even return verifiable optimal solutions.

2 Dubins curves

In general, this thesis will focus on solving a routing problem for autonomous agricultural vehicles on agricultural fields. As described in section 1, this includes computing the edges one has to take in order to connect all required edges that the vehicle has to perform labour on. These edges will mostly be referred to as 'turns'. Thus, one starts with computing these turns. Therefore, this section will introduce the notion of Dubins curves, a mathematical method of computing the shortest curve that connects two nodes in the two-dimensional Euclidean plane, with a constraint on the turning radius of the vehicle and with prescribed initial and final tangent. Furthermore, the assumption is made that the vehicle can only travel forwards. In practice, the vehicle is also not allowed to travel over already processed field in these turns nor leave the boundary of the agricultural field. However, this is something that is not accounted for in the computation of these turns.

2.1 An introduction to Dubins curves

Firstly, when using the notion of Dubins curves, one starts with the premise that a vehicle can essentially only do one of three things: turn right at maximum curvature (R), turn left at maximum curvature (L) or move straight forward (S). It has been proven by Lester Dubins, in 1957, that any shortest path made by such a vehicle as described will be made by joining circular arcs of maximum curvature and straight line segments [7]. Moreover, this result has been proven again in 1974 by H. Johnson [12], as well as by J.D. Boissonnat et al. in 1992 [5], both using Pontyagrin's maximum principle.

Furthermore, Dubins concluded in his paper, that a shortest path can always be expressed as a combination of 3 motion primitives, where the symbols R, L and S are the motion primitives as described above. Thus, using these symbols, any possible shortest path can be described using 3 symbols that correspond to the order in which these motion primitives are applied. Trivially, there is no need for two consecutive primitives to be identical, as these can be merged into one. Under this observation, Dubins has proven that only 6 of these combinations of symbols can possibly be optimal, namely:

$$\{RSR, LSL, RSL, LSR, RLR, LRL\} \tag{1}$$

Now, the shortest path between any two nodes in the two-dimensional Euclidean plane can always be characterized by one of the sequences described in (1), which are called the Dubins curves. Moreover, these 6 possible sequences can be compressed to 2 by introducing the symbol C that stands for 'curve', representing either the symbol R or L. Using the symbol C, the 6 sequences can be compressed to the following 2 sequences:

$$\{CSC, CCC\} \tag{2}$$

Note that this compressed form is defined under the assumption that two consecutive C's in a sequence must denote distinct symbols.

Now, in order to visualize the notion of these 2 possible types of Dubins curves, an example of such curves is depicted in figure 2, retrieved from [13]. More specifically, this figure displays an RSL and an RLR Dubins curve, including their respective initial and final configurations and the angles at which the turns are made along its turning radius.

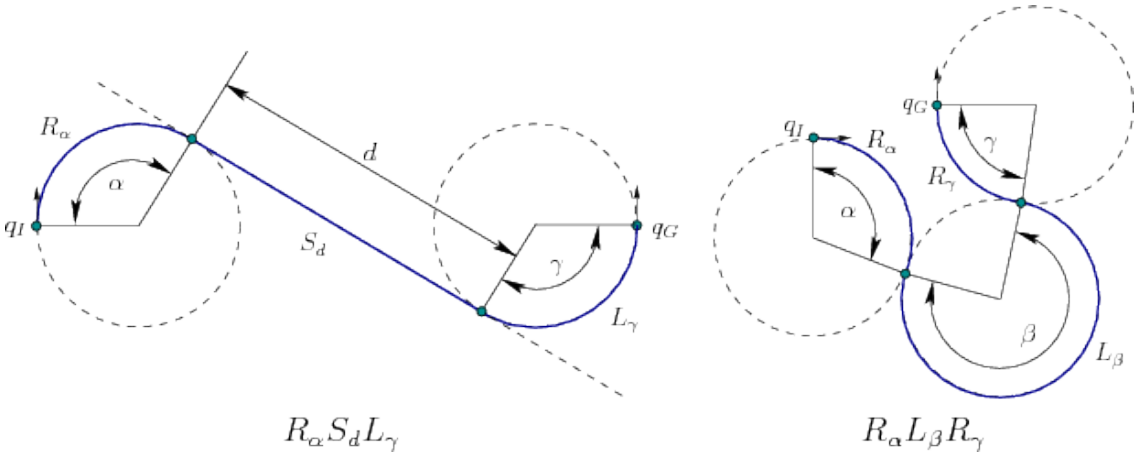


Figure 2: A visualization of an RSL and an RLR Dubins curve

2.2 Computing Dubins curves

Now that some of the basic principles of the Dubins curves, which will be used as turns in this thesis, have been explored, there needs to be a way of computing them. These turns will not be computed by hand, but rather by using a Python package [19], which uses an approach based on the algebraic solutions published in a paper written by A.M. Shkel et al. [17]. However, instead of using the classification scheme found in this paper, one has opted to use a more simple approach of testing all 6 possible solutions at the cost of some performance time.

Given a path from the initial to the final configuration, any point on this path is fully described by its Cartesian location $x(t), y(t)$, where the variable t can be interpreted as either the time or the length of the path traversed from the initial point P_i with unit velocity. Furthermore, assume

- the point on the path can only move forward from the initial point P_i towards the final point P_f
- one always moves at the unit speed
- the direction of motion cannot change faster than $\frac{1}{\rho}$, where without loss of generality $\rho = 1$ is the minimum turning radius

In this setting, any possible path is of the form described in (2) or a subpath of any of these path types, where the C in this case denotes an arc along a circle of radius 1. Moreover, the list of possible paths forms a sufficient set of optimal paths.

To specify these possible paths, the three motion primitives introduced earlier (L, R, S) will be used. Furthermore, three corresponding operators will be needed, namely L_v, R_v and S_v, which transform an arbitrary point $(x, y, \phi) \in \mathbb{R}^3$ into its corresponding image point in $\mathbb{R}^3$, where the index v indicates the motion to be of length v along a segment (C or S). These operators are defined as follows:

$$\begin{aligned} L_v(x, y, \phi) &= (x + \sin(\phi + v) - \sin(\phi), y - \cos(\phi + v) + \cos(\phi), \phi + v), \\ S_v(x, y, \phi) &= (x + v\cos(\phi), y + v\sin(\phi), \phi), \\ R_v(x, y, \phi) &= (x - \sin(\phi - v) + \sin(\phi), y + \cos(\phi - v) - \cos(\phi), \phi - v). \end{aligned} \tag{3}$$

Using these operators, any path in the set of Dubins curves described in (1) can be expressed using the corresponding equations. Without loss of generality, the initial configuration of each is at $(0, 0, \alpha)$ and the final configuration at $(d, 0, \beta)$, where d denotes the Euclidean distance between the initial and final point. For example, an LRL path, with segment lengths t, p and q respectively,

which starts at $(0,0,\alpha)$, has to end at $L_q(R_p(L_t(0,0,\alpha))) = (d,0,\beta)$, where the length of the path, D can simply be definied as the sum of its segments:

$$D = t + p + q. \tag{4}$$

For illustrative purposes, one of each compressed form of the Dubins curves described in (2) will now be considered, and the operator equations for the length of each path will be derived.

1. *RSR*: This means that $R_q(S_p(R_t(0,0,\alpha))) = (d,0,\beta)$. Using (3), one obtains the following system of scalar equations:

$$\begin{aligned} p\cos(\alpha - t) + \sin(\alpha) - \sin(\beta) &= d, \\ p\sin(\alpha - t) - \cos(\alpha) + \cos(\beta) &= 0, \\ \alpha - t - q &= \beta \mod 2\pi. \end{aligned}$$

The solution of this system of scalar equations yields the length of the corresponding segments, namely:

$$\begin{aligned} t_{rsr} &= \alpha - \arctan(\frac{\cos(\alpha) - \cos(\beta)}{d - \sin(\alpha) + \sin(\beta)}) \mod 2\pi, \\ p_{rsr} &= \sqrt{2 + d^2 - 2\cos(\alpha - \beta) + 2d(\sin(\beta) - \sin(\alpha))}, \\ q_{rsr} &= -\beta + \arctan(\frac{\cos(\alpha) - \cos(\beta)}{d - \sin(\alpha) + \sin(\beta)}) \mod 2\pi. \end{aligned} \tag{5}$$

Now implementing (5) into (4) yields:

$$D_{rsr} = t_{rsr} + p_{rsr} + q_{rsr} = \alpha - \beta + p_{rsr}. \tag{6}$$

2. *RLR*: This means that $R_q(L_p(R_t(0,0,\alpha))) = (d,0,\beta)$. Using (3), one obtains the following system of scalar equations:

$$\begin{aligned} 2\sin(\alpha - t + p) - 2\sin(\alpha - t) &= d - \sin(\alpha) + \sin(\beta), \\ -2\cos(\alpha - t + p) + 2\cos(\alpha - t) &= \cos(\alpha) - \cos(\beta), \\ \alpha - t + p - q &= \beta \mod 2\pi. \end{aligned}$$

The solution of this system of scalar equations yields the length of the corresponding segments, namely:

$$\begin{aligned} t_{rlr} &= \alpha - \arctan(\frac{\cos(\alpha) - \cos(\beta)}{d - \sin(\alpha) + \sin(\beta)}) + \frac{p_{rlr}}{2} \mod 2\pi, \\ p_{rlr} &= \arccos(\frac{1}{8}(6 - d^2 + 2\cos(\alpha - \beta) + 2d(\sin(\alpha) - \sin(\beta)))), \\ q_{rlr} &= \alpha - \beta - t_{rlr} + p_{rlr} \mod 2\pi. \end{aligned} \tag{7}$$

Now implementing (7) into (4) yields:

$$D_{rlr} = t_{rlr} + p_{rlr} + q_{rlr} = \alpha - \beta + 2p_{rsr}. \tag{8}$$

Likewise, the other 4 Dubins curves described in (1) can be examined, deriving the operator equations for the length of each path. The Python package used will do this as well for all 6 possible Dubins curves and pick the shortest one, which, as stated earlier, was proven by Dubins [7] to be optimal.

3 A heuristic algorithm

In this section, the current solution to the routing problem, based on a heuristic algorithm, will be presented. Furthermore, some problems that arise using this algorithm will be discussed and these problems will be addressed as well.

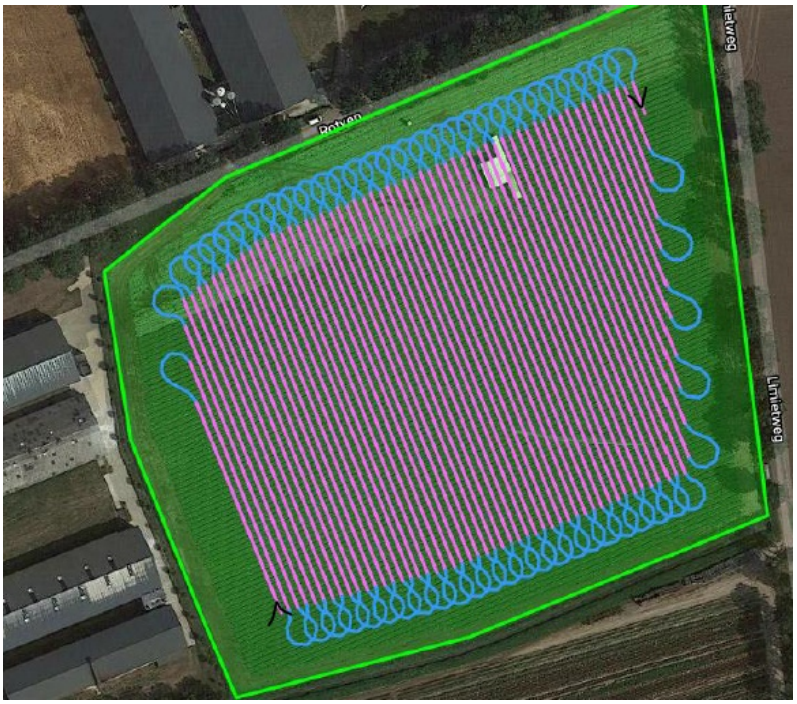
3.1 The current solution

Currently, the company Phact is using a solution to the routing problem based on the heuristic algorithm based on Monte Carlo methods presented in a paper by P. Fernández De Córdoba et al. [8]. This algorithm was implemented by L. Philipse [15] in her Bachelor thesis performed for the company.

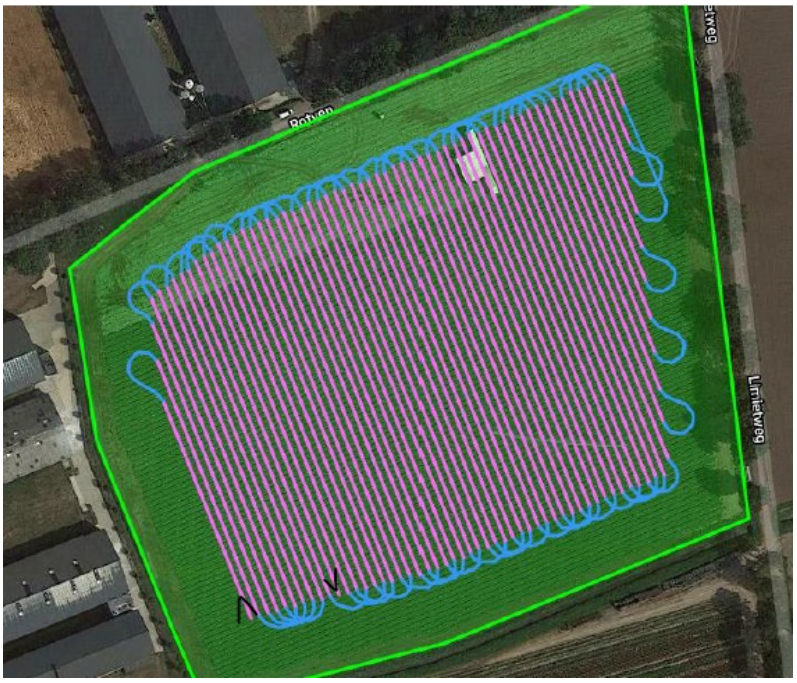
In general, the idea of the approach of the algorithm currently being used is to simulate a vehicle traveling randomly over a graph for a predetermined number of iterations. Every iteration, the algorithm starts by adding a random unvisited node to a list. Then, by definition, the corresponding required edge is not yet traversed and will thus be traversed. From here, the node on the other end of the required edge will be added to the list of visited nodes and the algorithm will select any reachable unvisited node randomly, based on probabilities calculated by the algorithm, which for instance take into account the cost of an edge, an edge either being required or not and a node either being required or not. This process repeats itself until all required edges are traversed. A specified number of routes is examined by the algorithm, where the shortest one is considered to be the output.

In comparison to other algorithms solving the Rural Postman Problem, as described in section 1, this algorithm distinguishes itself from them by its simplicity and adaptability. As such, one can alter the number of iterations performed or the definition of the probabilities used by the algorithm to either obtain very fast and feasible results, or obtain very accurate results. Furthermore, one would only have to make small changes to the algorithm to deal with other Postman Problems in which additional constraints are presented.

As shown in [8], on the input of a good choice of parameters, this algorithm has proven to perform very well in many instances. Moreover, the algorithm described above has also been implemented on some agricultural fields provided and has shown to provide very significant improvements to the problem at hand, in comparison to naive solutions, as described in section 1. This improvement can quite clearly be seen in a comparison between the two, which has already been implemented by the company, depicted in figure 3.



(a) A naive route on the main land



(b) Implementation of the algorithm

Figure 3: Comparison of a naive solution to that of the algorithm

These results all seem very encouraging, and to some extent they certainly are, however there are some issues to take note of as well. Firstly, as one might have noticed, the algorithm finds a route by starting at a random node, as well as ending at a random node. This might not seem to be of great importance, however, in practice it is very desirable for these points to be determined by

the user. Furthermore, in the case of these agricultural fields, the algorithm has shown to provide significant improvements on the routing with respect to naive solutions, however there is at this point no objective measure against these results to show exactly how good they are and how much room for improvement there is. The latter is something to be explored later in this thesis.

3.2 An improvement to the current solution

In practice, as described in section 1, one wants to start from some point on the boundary of an agricultural field to go to a point on the main land, let the autonomous agricultural vehicle perform its labour on the route on the main land, then perform its labour on the head land inside out, and lastly end at the same point on the border where one started, so that the vehicle can be removed from the field without negating any work done. As described, this is not currently the case, as both the starting and end point on the main land are currently randomly chosen. Furthermore, the current algorithm also does not take into account the distance that has to be traveled from the border of an agricultural field to the starting point of the route. This can be very expensive, as can for instance be seen in figure 4, which is an implementation of the algorithm currently being used. In this implementation the starting edge connecting the starting point on the boundary to the main land is the most expensive turn in the entire route. To be more precise, the cost of the edge connecting the starting point on the boundary to the main land is 186.33 out of a total cost of the turns of 1234.07, meaning that it makes up 15.10% of the total cost of the turns on the route. Lastly, it is also important to note that turns in general cannot cross the land that has already been worked on by the vehicle. However, if this has not yet been done this is possible and will sometimes be the case for the edge that connects the starting point on the boundary of the field to the main land, as can be seen in figure 4.

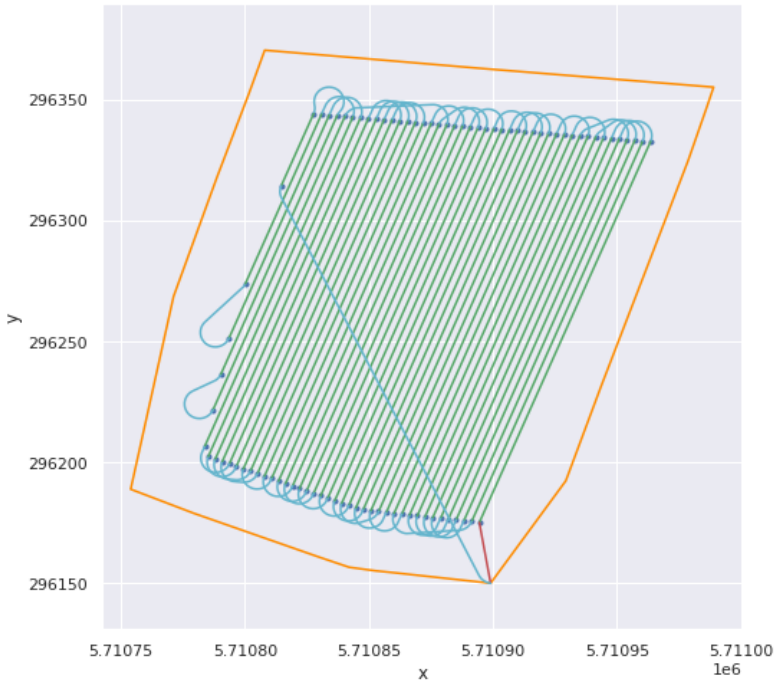


Figure 4: Example solution for field 1 of cost 1234.07 using the current algorithm

As the algorithm is quite adaptable, the company Phact has already implemented a fixed starting point on the main land in the current algorithm. The way this is currently being used is such that one starts at the desired starting point and then uses the heuristic algorithm described to plan a route on the main land. Then one inverts the route and closes it by adding a final edge. This

essentially fixes the end point on the main land, meaning that it is known to the user where the vehicle will end up at on the boundary of the field after completing its route around the polygons on the head land, which is also displayed in figure 4 by the red line.

Now, this is quite a good improvement of the algorithm already, however another problem remains present. Namely, one still does not know beforehand where on the main land the route will actually start and the distance that has to be traveled from the starting point on the border to this point is not taken into account in the cost of the route. The first of these two problems will probably appear to be of greater concern than it is in actuality, however the second might skew the results in a negative manner to quite a large extent. Therefore, a slight alteration to the algorithm has been made, in which, given a fixed starting configuration (Cartesian coordinates and tangent) on the border of an agricultural field, the algorithm now also takes into account the Dubins curves between the starting point on the boundary and the main land. This is a quite trivial and simple fix to the problems described, however it does solve them and as will be shown now, the results are also quite significant.

3.3 Results

In this section, the results that are obtained from the improvements and additions that have been made, described in section 3.2, will be displayed and discussed with respect to the results obtained from the current algorithm. The improved algorithm can be found in appendix A.1. These results will be analyzed with respect to 3 agricultural fields that were provided and based on 1,000 iterations of the respective algorithms, as well as other parameter values being equal, in order to give an objective review. Moreover, these fields were also analyzed in [15]. Note that the lengths of the routes displayed only concern the turns that are made on the route and do not include the lengths of the required edges, as these are fixed and would not mean anything to the results. Furthermore, also note that the red line in these figures has a cost of 0, and is only there for illustrative purposes, displaying the change between the start and end position of the vehicle when performing its route on the head land inside out.

Firstly, field 1 will be considered, depicted in figure 4. As mentioned before, the total cost of the turns on this route, using the current algorithm, is 1234.07. Now, using the algorithm with the described alterations, this yields a solution with the total cost of the turns being 1082.06. This solution is displayed in figure 5. This means that the new solution is 12.32% cheaper than the solution found by the algorithm currently being used.

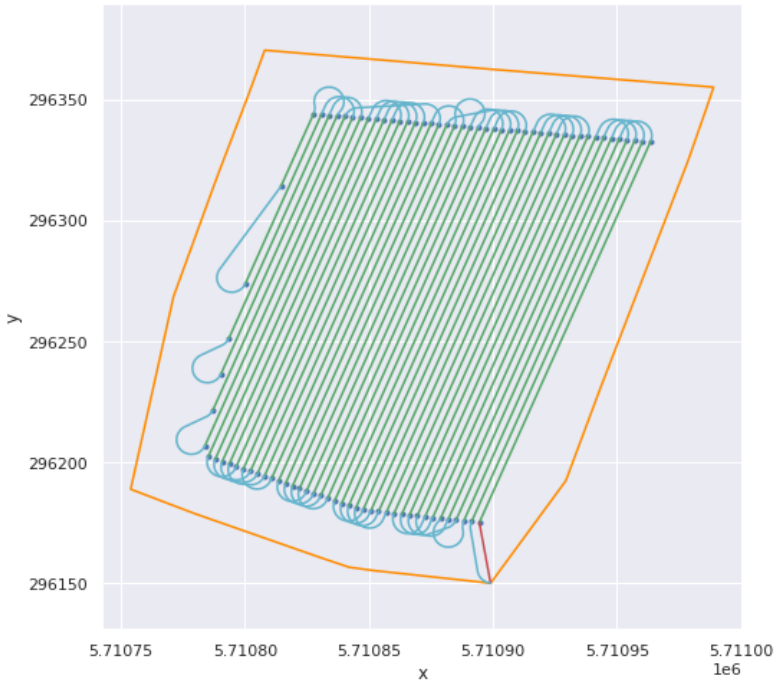
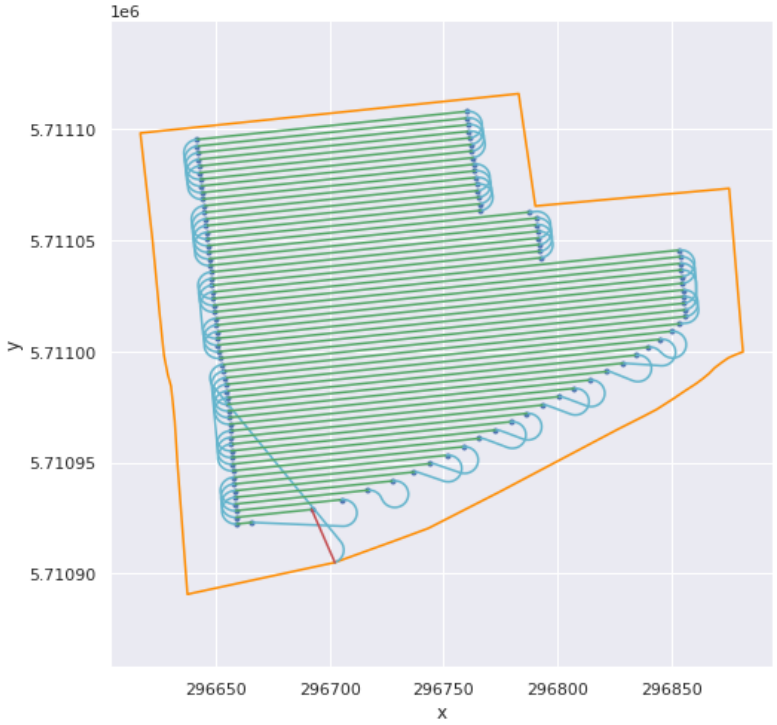


Figure 5: Example solution for field 1 of cost 1082.06 using the improved algorithm

Secondly, field 2 will be considered, which is of a somewhat more irregular shape. The solution routes to this field using both the current and the new improved algorithm are depicted in figure 6. From these solution costs, one can conclude that the new improved algorithm provides a solution that reduces the costs by 3.67% with respect to the solution found by the current algorithm. This is a significant decrease in cost, however it is quite a considerably smaller decrease than was achieved on field 1.



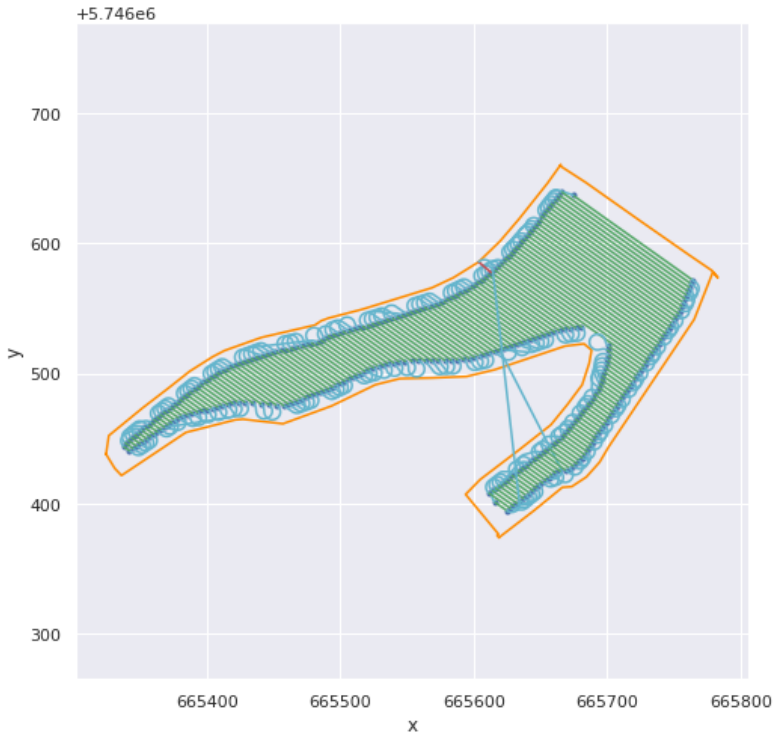
(a) Example solution of field 2 of cost 1496.99 using the current algorithm

(b) Example solution of field 2 of cost 1442.12 using the improved algorithm

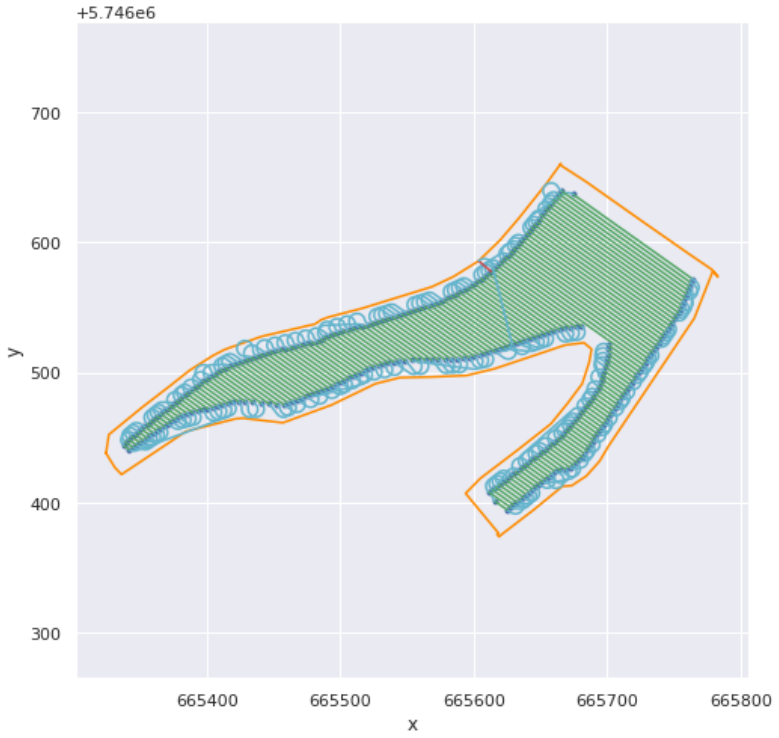
Figure 6: Comparison of the current and improved algorithm on field 2

Lastly, field 3 will be analyzed, which is a larger field of irregular shape, which might cause some

issues. It is important to note that a field like this is certainly an outlier, however it is interesting to see how the algorithm performs on a field like this and how things change in general when analyzing such fields. The example solutions to this field using both the current and the improved algorithm are displayed in figure 7. From the costs of the routes created by both algorithms one can conclude that the improved algorithm reduces the costs by 1.98% with respect to the current algorithm. This is again a smaller decrease in cost in comparison to the previous field, however still a significant cost reduction. Furthermore, it is important to note that the algorithm does run into some difficulties when dealing with a field like the one depicted in figure 7. This can be seen by the fact that some turns leave the boundaries of the field, which cannot happen in practice. In these example solutions it only seems to happen using the current algorithm. However, after some testing it turns out that this can happen to the improved algorithm as well. Although, it seems to occur less often, which might be explained by the fact that this algorithm at least forces, to some extent, the final point on the route to be close to the starting point on the boundary of the field. Moreover, when the turns that run outside of the boundaries of the field would be compensated for, the new algorithm should turn out to be more of an improvement than it is right now.



(a) Example solution of field 3 of cost 4551.39 using the current algorithm



(b) Example solution of field 3 of cost 4461.33 using the improved algorithm

Figure 7: Comparison of the current and improved algorithm on field 3

A summary of these results can be seen in table 1.

Field	Number of edges	Old algorithm	Improved algorithm
1	44	1234.07	1082.06
2	59	1496.99	1442.12
3	169	4551.39	4461.33

Table 1: Table containing the results obtained thus far

In conclusion, one might notice that the relative improvements made by the new algorithm with respect to the current algorithm grow smaller for every consecutive field in these examples. This could be explained rather simply by the fact that the impact of the alterations that have been made to the current algorithm in order to create the new algorithm only affect the cost of one edge. Thus, when the fields get larger and contain more turns, these effects will consequently decrease as well. This can be seen in the results as well, as the cost of the route on the main land without the cost of the edge connecting the boundary of the field to the main land remains the same to a very large extent in all cases and only one edge, which ranges from moderately to very expensive, becomes significantly cheaper. However, even though the effects of the improved algorithm decrease as the fields get bigger, it has shown to provide significant results in all cases, even for the rather large field 3, and would thus be a good implementation for the company.

4 1-tree Lagrangian relaxation

In this section, the goal is to find a tight lower bound (lb) for the routing problem at hand, in order to assess the results obtained so far and to determine how much room for improvement there is left.

4.1 Minimum Spanning Tree

One rather simple way to determine a lower bound for a problem such as this, is to use the notion of a Minimum Spanning Tree (MST). This notion will be introduced along some graph related definitions:

Definition 1. A forest is a graph without circuits.

Definition 2. A tree is a connected forest.

Definition 3. $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$ if G' is a graph and $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$, then G' is called a spanning subgraph.

Definition 4. A spanning subgraph $T = (V, F)$ of a graph $G = (V, E)$ is called a spanning tree of G if T is a tree.

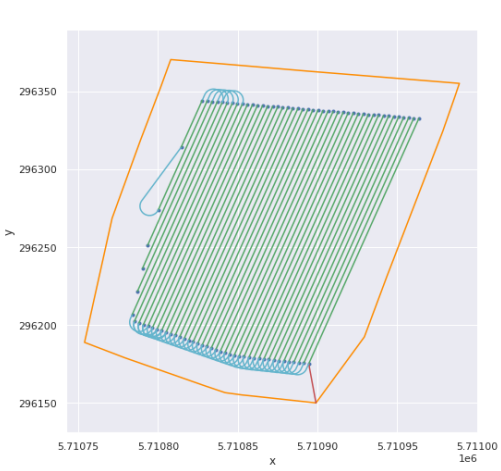
Now, given a weight function on the edges of a graph, one can conclude from these definitions that a MST is a spanning tree of minimum weight, or in other words, a MST is a subset of the edges of a connected graph that connects all the vertices, without any cycles and at minimum cost. Adding some slight manipulations to the weight function on the edges of the graph, such a MST would be a lower bound for the routing problem at hand, as it is the cheapest way to connect all the vertices of the graph. More specifically, this is the case as these manipulations will also ensure the required edges to be contained in the MST, as will be seen later.

Now, there are multiple algorithms to compute MSTs and in this case it has been chosen to use Kruskal's algorithm, which has a running time $\mathcal{O}(E \log(V))$ [6]. Essentially, the algorithm continuously adds the cheapest possible edge without closing any circuits until this is no longer possible, resulting in a minimum spanning tree. The slight manipulation used to ensure that all required edges will be contained in this MST and to ensure that this can actually be used as a lower bound is to give all required edges a weight of zero, which will result in this edges being picked first after which the algorithm continues to add the turns without closing any circuit. Lastly, an addition made to the algorithm will be to close the MST by connecting the tree in the cheapest possible manner to the starting point on the boundary of the field. This algorithm is described in pseudo-code in algorithm 1

Algorithm 1 Kruskal's Algorithm, with closing

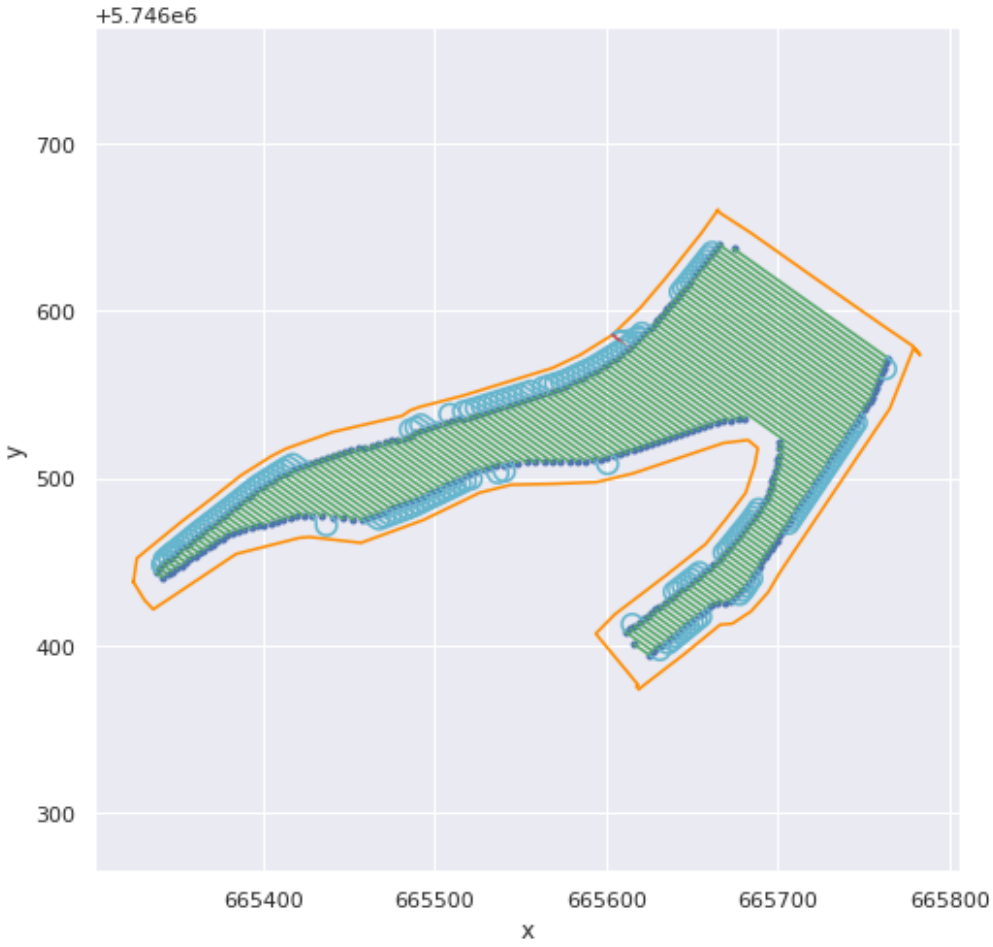
1: Input: A weighted graph $G = (V, E)$, where V represents the set of nodes and E the required edges and turns
2: $F := \emptyset$
3: **for** $k = 1, 2, \cdots, |V| - 1$ **do**
4: Pick an edge $e_k \in E \backslash F$ of minimal length, such that $F \cup \{e_k\}$ is a forest
5: $F := F \cup \{e_k\}$
6: **end for**
7: Connect the starting point on the boundary to the two points closest in F

Now, this has been implemented in Python, using the package NetworkX [1], which has a built in function to perform Kruskal's algorithm. The results of this implementation can be seen in figure 8.



(a) MST of field 1 with cost 974.45

(b) MST of field 2 with cost 1160.00



(c) MST of field 3 with cost 3817.32

Figure 8: MSTs of all fields connected to the boundary of the field

Now, using these MSTs connected to the boundary of the fields depicted in figure 8, one has obtained some lower bounds for the given fields. Using these values, one can analyze the quality of

the results obtained by the algorithm described in section 3. For illustrative purposes, all results obtained thus far are presented in table 2. These computations yield that, at best, for the fields 1, 2 and 3, the costs can be reduced 9.94%, 19.56% and 14.65% respectively. From this one might conclude that there is still a lot of room for improvement and that the algorithm described in section 3 does not perform too well. However, one has to note that this is just a lower bound and not necessarily a tight or good one. To illustrate this, one could look at the figures itself and note that these do not even remotely resemble a route. To be more specific, a large majority of the nodes in these MST's are not of degree 2, meaning that they cannot form a circuit as one would like as a solution for the routing problem. In detail, this means that for field 1, 75 out of 89 nodes are not of degree 2, for field 2 this are 97 out of 119 nodes and for field 3, 252 out of 339.

Field	Number of edges	Old algorithm	Improved algorithm	MST
1	44	1234.07	1082.06	974.45 (lb)
2	59	1496.99	1442.12	1160.00 (lb)
3	169	4551.39	4461.33	3807.59 (lb)

Table 2: Table containing the results obtained thus far

Using the method described, a rather simple lower bound has been computed. However, this method has shown to provide a lower bound that might be far from a real optimal solution. Thus, one would like to find a more tight lower bound that comes closer to represent a real solution, and thus also comes closer to the value of a real optimal solution. This will be explored in the following sections, using the notion of minimum spanning trees as well.

4.2 Relaxation

Now, the notion of relaxation will be introduced, which is a modeling strategy in the field of mathematical optimization. Moreover, this general introduction into relaxation will later be expanded to Lagrangian relaxation.

A relaxation is an approximation of a difficult problem by a closely related problem that is easier to solve. A solution to the relaxed problem provides information regarding the original problem. In this case, this information would include a lower bound to the original problem at hand.

To illustrate this principle, consider the following relaxation. A relaxation of the minimization problem

$$z = \min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$$

is another minimization problem of the form

$$z_R = \min\{c_R(x) : x \in X_R \subseteq \mathbb{R}^n\}.$$

This comes with the following two properties:

1. $X_R \supset X$
2. $c_R(x) \leq c(x)\ \forall x \in X$.

The first property states that the set of feasible solutions X_R properly contains all feasible solutions X. Furthermore, the objective function of the relaxed problem is an arbitrary extension on the set of feasible solutions X_R of the objective function of the original problem. Consequently, the second property follows, stating that the objective function value of an optimal solution to the relaxed problem is less than or equal to the objective function value of an optimal solution to the original problem [10]. To illustrate this, suppose that x^* is an optimal solution to the original problem. Then, following the first property, $x^* \in X \subset X_R$ and $z = c(x^*) \geq c_R(x^*) \geq z_R$. Therefore, an optimal solution to the original problem provides an upper bound on the relaxed

problem. Furthermore, removing a set of constraints from a minimization problem can only reduce the objective function value. Thus, if $\hat{x}$ is an optimal solution to the relaxed problem, then $z_R = c_R(\hat{x}) \leq c_R(x^*) \leq c(x^*) = z$ and one can conclude that the optimal value solution to the relaxed problem provides a lower bound on the optimal value of the original problem. Moreover, if in addition to these properties, $c_R(x) = c(x), \forall x \in X$, the following holds: if an optimal solution for the relaxed problem is a feasible solution for the original problem, then it is optimal for the original problem [9].

4.3 Defining the original problem

Until now, the problem at hand has mostly been defined as the Rural Postman Problem. In this section, this will be elaborated on, with respect to a feasible set of solutions, in order to be used in a relaxation.

The solution set of the problem at hand is the set $\mathcal{H}_n$ of all Hamiltonian cycles of K_n, which is a complete graph on n vertices [10]. Moreover, the solution needs to contain all required edges. Now, a Hamiltonian cycle which is part of the set of feasible solutions is a subgraph $H = (V_n, E)$ of K_n satisfying the following requirements:

a) All nodes of H have degree 2;

b) H is connected;

c) H contains all required edges.

Now that the original problem is defined properly, with a set of feasible solutions, it remains to determine a relaxation of this problem whose set of feasible solutions properly contain this set of feasible solutions.

4.4 1-tree relaxation

A relaxation of the problem at hand can be obtained by defining a problem whose feasible solutions are subgraphs of K_n, satisfying a subset of the requirements described in section 4.3 and whose optimal solution can be found in polynomial time. These requirements imply that any subgraph of K_n that satisfies them has exactly n edges and spans K_n, therefore one only has to consider relaxations having as feasible solutions spanning subgraphs of K_n with n edges.

Firstly, one can obtain a relaxation by dropping requirement (a) from section 4.3 for all nodes except one, without loss of generality let this be node 1. Now, denote by $K_n \setminus \{1\}$ the subgraph of K_n induced by $V_n \setminus \{1\}$. The subgraph of a Hamiltonian cycle induced by the nodes in $V_n \setminus \{1\}$ is a Hamiltonian path of $K_n \setminus \{1\}$. This is a subgraph $\mathcal{H}_p$ of $K_n \setminus \{1\}$ satisfying the requirements [10]:

a) $\mathcal{H}_p$ spans $K_n \setminus \{1\}$;

b) $\mathcal{H}_p$ has $n - 2$ edges;

c) $\mathcal{H}_p$ is connected;

d) all nodes of $\mathcal{H}_p$ have degree 2, except two.

Now, obviously the union of a Hamiltonian path on $K_n \setminus \{1\}$ and of two edges incident with node 1 is a relaxation of a Hamiltonian cycle. However, this is not a particularly useful relaxation, as finding a Hamiltonian path in $K_n \setminus \{1\}$ is as difficult as finding a Hamiltonian cycle in K_n. On the other hand, instead of using a Hamiltonian path, one can also consider a relaxation obtained by dropping the requirement mentioned above of being a Hamiltonian path and use the resulting graph, which is a tree that spans $K_n \setminus \{1\}$.

Using this, one can produce a useful relaxation of the original problem by taking the union of a tree that spans $K_n \setminus \{1\}$ and a pair of edges incident with node 1. Such a graph will be denoted

as a 1-tree. Now, the 1-tree that minimizes the objective function c can be obtained by finding the MST in $K_n \setminus \{1\}$ and connecting to it the two shortest edges incident with node 1. Thus, the complexity of finding a minimum 1-tree, which is the relaxation, is the same as that of finding a MST, which is polynomial as mentioned in section 4.1.

In order to strengthen the lower bound obtained by this relaxation, one could modify the objective function. If, $\forall v \in V_n$ one adds a constant λ_v to the objective function coefficients of all edges of $\delta(v)$ (the set of all edges incident with v), the length of all Hamiltonian cycles increase by the same amount $\sum_{v \in V_n} 2\lambda_v$. Now, for any edge (u, v) of K_n, the coefficient of the new objective function is $c'(u, v) = c(u, v) + \lambda_u + \lambda_v$, as (u, v) belongs to both $\delta(u)$ and $\delta(v)$. Consequently, the optimal value of the original problem

$$\min_{H \in \mathcal{H}_n} \{\sum_{i<j} (c_{i,j} + \lambda_i + \lambda_j) x^H_{i,j}\} - 2 \sum_{i \in V} \lambda_i$$

does not depend on the vector $\lambda \in \mathbb{R}^{V_n}$ [10]. However, this does not generally hold for 1-trees, as these do not often hold the requirement of having all nodes of degree 2. Therefore, the length of an optimal 1-tree

$$L(\lambda) = \min_{x^{1T}} \{\sum_{i<j} (c_{i,j} + \lambda_i + \lambda_j) x^{1T}_{i,j}\} - 2 \sum_{i \in V} \lambda_i$$

is a function of $\lambda \in \mathbb{R}^{V_n}$, where x^{1T} is the incidence vector of a 1-tree in K_n [10]. Now, $L(\lambda)$ is a lower bound for the original problem and it generally yields different lower bounds for different vectors λ. Thus, the tightest bound is obtained by solving the maximization problem

$$\max_{\lambda} \{L(\lambda)\},$$

which is called the Lagrangean dual problem and the bound obtained by solving this problem is known as the Held-Karp bound [11].

4.5 Implementation

In this section, an implementation of the method of determining lower bounds is discussed. This method is based on the Lagrangean dual problem, based on the 1-tree relaxation described in section 4.4. Note that the corresponding Lagrangean dual problem is of the form $\max_\lambda \{L(\lambda)\}$. In order to determine the value of $L(\lambda)$ for a given vector of node multipliers λ, one has to compute a 1-tree with respect to the modified edge weights $c(i, j)+\lambda_i+\lambda_j$ and subtract the surplus $2\sum_{i=1}^{n} \lambda_i$.

Now, a method used for maximizing $L(\lambda)$ is the subgradient method. This method is an iterative gradient ascent method, where in each iteration the next vector of node multipliers is determined by taking a step in the direction of a subgradient. In the minimum 1-tree a subgradient is supplied as follows. Let δ_i be the degree of node i in the minimum 1-tree with respect to node multiplier λ. Then the vector $(\delta_1 - 2, \delta_2 - 2, \cdots, \delta_n - 2)$ is a subgradient of L at λ. More specifically, the vector λ is often referred to as the Lagrange multiplier, which imposes a cost on violations of the constraints that are present, as can be seen here for nodes of degree other than 2. Essentially, this means that the Lagrange multiplier will make, on the one hand, any node of degree 1 more appealing to be visited and, on the other hand, nodes of a degree higher than 2 less appealing. Moreover, this inclusion of the Lagrange multiplier makes the relaxation method for this problem a Lagrangian relaxation. Now, if the function $L(\lambda)$ is bounded from above (which it is in the RPP), and if the step lengths α_k satisfy both $\lim_{k\to\infty} \alpha_k = 0$ and $\sum_{k=0}^{\infty} \alpha_k = \infty$ then the subgradient method is proven to converge to the maximum of L, as desired [16]. However, in practice it has turned out that such step length formulas lead to very slow convergence. Moreover, there are step length formulas that do not satisfy the requirement that $\sum_{k=0}^{\infty} \alpha_k = \infty$, but lead to better convergence in practice.

Thus, a different implementation of a method solving the Lagrangean dual problem has been used, based on [4], [18], [10] and the subgradient method, described in algorithm 2. In this algorithm, a weighted combination of the subgradients d^k and d^{k-1} has been used, as can be seen in line 9 of the algorithm.

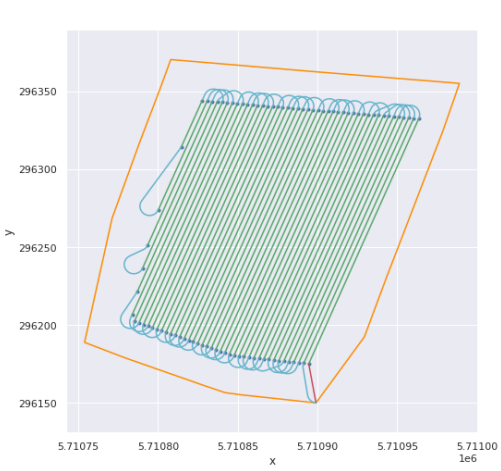
Algorithm 2 Lagrangian 1-tree relaxation $(\alpha_1, \gamma, k_{max}, f_{max}, G)$

1: Input: Complete weighted graph with required edges G, a maximum number of iterations k_{max} and a maximum number of consecutive fails without improvement f_{max}.
2: Let α_1 be the initial step length and γ a decrement factor for this step length.
3: Set $\lambda_i^1 = 0$ for every node i, $k = 1$ and $f = 0$.
4: **while** $k < k_{max}$ AND $f < f_{max}$ **do**.
5: Compute a minimum spanning tree with respect to the edge weights $c_{i,j} + \lambda_i + \lambda_j$.
6: Compute a minimum 1-tree from this minimum spanning tree and its length.
7: Check whether or not the bound obtained from the minimum 1-tree is higher than the current best bound and if so set $f = 0$, else increment f by 1.
8: Compute the vector d^k, where $d_i^k = \delta_i - 2$, where δ_i is the degree of node i in the 1-tree computed in the previous step.
9: Compute the new vector λ^{k+1} where $\lambda_i^{k+1} = \lambda_i^k + \alpha_k(0.7d_i^k + 0.3d_i^{k-1})$ for every node i and $d_i^0 = 0$.
10: Set $a_{k+1} = \gamma a_k$ and increase k by 1
11: **end while**
12: Return the best 1-tree computed with its corresponding lower bound

4.6 Results

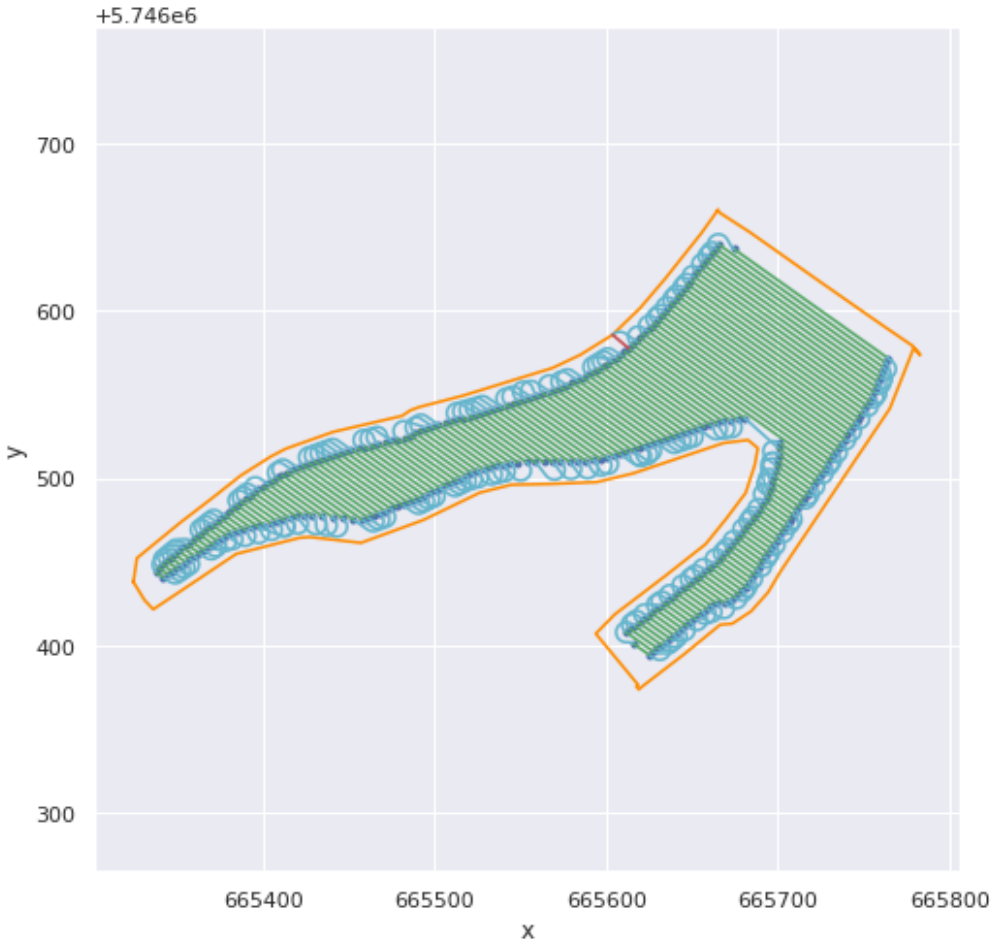
In this section, the results obtained from implementing algorithm 2 will be presented and discussed. The algorithm was implemented in Python and the code can be found in appendix A.2.

In order to analyze the results obtained from this algorithm, it will again be implemented on the same three agricultural fields. The results of this implementation can be seen in figure 9, displaying the 1-trees obtained from this implementation.



(a) 1-tree of field 1 with cost 1059.09

(b) 1-tree of field 2 with cost 1366.82



(c) 1-tree of field 3 with cost 4054.00

Figure 9: 1-trees of the three agricultural fields

Furthermore, all results obtained thus far are presented in table 3.

Field	Number of edges	Old algorithm	Improved algorithm	MST	1-tree
1	44	1234.07	1082.06	974.45 (lb)	1059.09 (optimal)
2	59	1496.99	1442.12	1160.00 (lb)	1366.82 (optimal)
3	169	4551.39	4461.33	3807.59 (lb)	4054.00 (lb)

Table 3: Table containing the results obtained thus far

Firstly, regarding field 1, the lower bound obtained by the algorithm was found after 682 iterations, after which no further improvements could be obtained. When looking at the 1-tree obtained in figure 9a, one might note that it is actually a Hamiltonian cycle containing all required edges, which are the requirements for a solution to the original problem. Now, as stated in section 4.2, if an optimal solution for the relaxed problem is a feasible solution for the original problem, then it is optimal for the original problem [9]. Thus, the lower bound obtained by the algorithm is actually, in this case, an optimal solution to the routing problem at hand with value 1059.09. Furthermore, this also allows one to assess the results obtained by the improved algorithm provided in section 3.2. When comparing the results presented in table 3, one can conclude that the solution obtained by the improved algorithm is only 2.37% more expensive than the optimal solution, from which one could conclude that this algorithm has also provided rather decent results on this field. Moreover, the implementation of the Lagrangian 1-tree relaxation has also enabled one to, in this case, increase the lower bound to the solution by 8.69%.

Secondly, as regards agricultural field 2, the lower bound found by the algorithm was obtained after 498 iterations, after which no further improvements were found. When looking at the 1-tree obtained in figure 9b, one might again note that it is actually a Hamiltonian cycle containing all required edges, which are the requirements for a solution to the original problem. Now, as stated in section 4.2, this again means that the lower bound obtained by the algorithm is actually, in this case, an optimal solution to the routing problem at hand of value 1366.82. Furthermore, one can also use this result to assess the results obtained by the algorithm described in section 3.2. When comparing the results presented in table 3, one can conclude that the solution obtained by the improved algorithm is 5.51% more expensive than the optimal solution, meaning that the costs of the improved algorithm could at best be reduced by 5.22%. This does leave quite some room for improvement of the algorithm described in section 3.2, however, it also shows that the algorithm does provide fairly decent results to some extent. Moreover, the implementation of the Lagrangian 1-tree relaxation has also enabled one to, in this case, increase the lower bound to the solution by 17.83%.

Lastly, regarding field 3, the lower bound found by the algorithm was obtained after 3056 iterations, where no further improvements to the lower bound could be found at least until 10000 iterations were reached. When looking at the 1-tree obtained in figure 9c, one might note that, in this case, it is not a Hamiltonian cycle. Thus, in this case, the algorithm is not able to find a solution that is actually an optimal solution to the original problem at hand as well. This is most likely the case due to the field being of too much of an irregular shape or of it simply being to big, as it contains more than three times the number of nodes as field 1 and close to 3 times the number of nodes as field 2. However, when comparing this result to the result obtained by the minimum spanning tree depicted in figure 8c, one can conclude that the lower bound has been increased by 6.47%. Moreover, one can observe from the figures itself as well that the 1-tree comes much closer to representing a solution to the original problem at hand, as only 50 out of the 339 nodes are of a degree other than 2, instead of the 252 out of 339 in the minimum spanning tree. Furthermore, this result also allows one to assess the results obtained by the algorithm provided in section 3.2. When comparing the results presented in table 3, one can conclude that the solution obtained by the improved algorithm is at most 10.05% more expensive than the optimal solution, meaning that the costs of the improved algorithm could at best be reduced by 9.13%. This does leave quite some room for improvement for the improved algorithm, however, it is also important to note that this is a lower bound that might not be possible to reach, as the lower bound still contains many

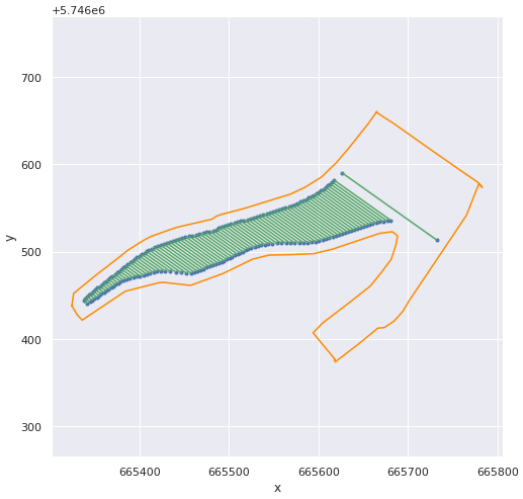
points that are not of degree 2, meaning that the actual optimal solution might be somewhat far away.

In conclusion, the Lagrangian 1-tree relaxation described in section 4.4 and implemented in section 4.5 does provide some very interesting results. Firstly, it has enabled one to find verifiable optimal solutions to agricultural fields 1 and 2. In general, it does seem to be able to find optimal solutions to fields of smaller size and more regular shape. Furthermore, it has also allowed one to improve the lower bounds found in section 4.1 which enables one to assess the results obtained by the improved algorithm described in section 3.2 to a greater extent. This has shown these results to be of fairly decent quality, yet leaving some room for improvement as well. Lastly, the algorithm was not able to find an optimal solution for field 3, which might be due to the size of the field or its irregular shape or even a combination of both, however it is important to note that this is generally a method to find tight lower bounds and not to find optimal solutions.

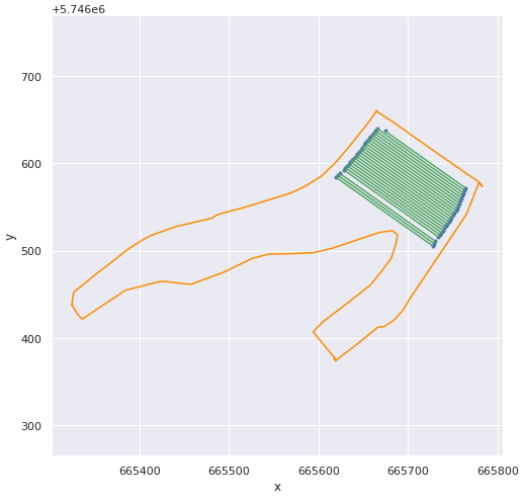
5 An ad hoc approach

In this section, an ad hoc approach to the problem at hand will be implemented, based on the information obtained thus far in this thesis. As could be seen in section 4.6, the Lagrangian 1-tree relaxation seemed to be able to find optimal solutions to smaller fields of a more regular shape. Now, the general idea behind this ad hoc approach will be that connecting some solutions that are piecewise optimal will result in a total connected solution that comes rather close to being optimal. Thus, in terms of the problem at hand, this means that the idea behind this approach will be to divide a field that cannot be solved optimally using the Lagrangian 1-tree relaxation, into some number of smaller fields of more regular shapes which the algorithm might be able to solve optimally and then connect these. Thus, one only continues to the next subfield when the previous one has been completely finished. In order to verify this idea and analyze its results, it will be implemented on field 3, depicted in figure 9c, which could not be solved optimally using the Lagrangian 1-tree relaxation.

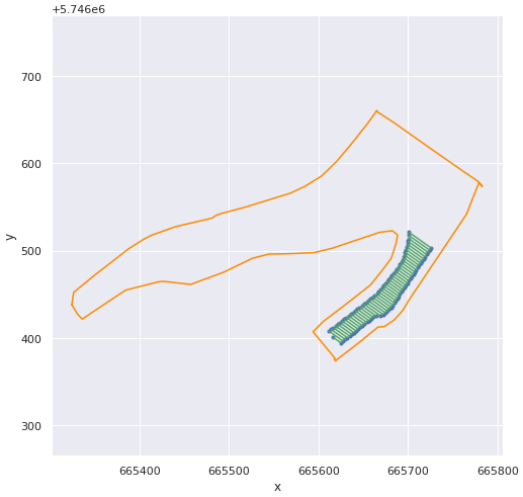
Firstly, the field has to be divided into smaller fields. In this case, it was chosen to divide field 3 rather intuitively into three fields, namely the left tail, the top part and the right tail, which will be called fields 3a, 3b and 3c respectively. This is depicted in figure 10. As can be seen from figures 10a and 10b, one required edge that is expected to be found in field 3b is included in field 3a. This is one of the manual modifications made to the fields in order to optimize the total length of the provided route. Adding this edge to field 3a enables one to cheaply connect it to field 3c, after which field 3b will be worked and the route will be finished. More specifically, in order to connect all smaller fields, the connecting edges have been chosen beforehand to ensure a lower cost. Moreover, it is important to note that the number of edges in fields 3a and 3c are even and this has been manipulated in this manner on purpose. Namely, this ensures that one finishes the route on the same side as one has started. Now, as the number of edges in field 3a are even, one will start at the boundary close to the top of field 3a, travel to the far side of field 3a and start the route on the main land there. This way, one can input this route to finish on the node on the top right, close to field 3c. This node will then be connected to field 3c and as the number of edges on field 3c are even as well, the route on field 3c will also finish on a node on the right-hand side. From here, this node will be connected to field 3b and as the number of edges in field 3b are odd, finishing the route on field 3b will mean that the complete route will finish on the opposite side again, namely the left-hand side near the starting point on the boundary of the field.



(a) Field 3a



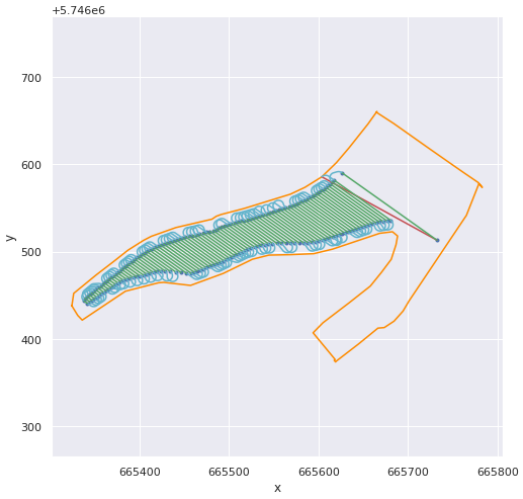
(b) Field 3b



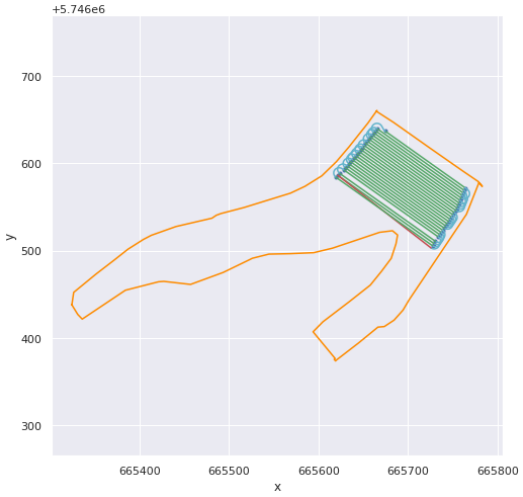
(c) Field 3c

Figure 10: Field 3 divided into three subfields

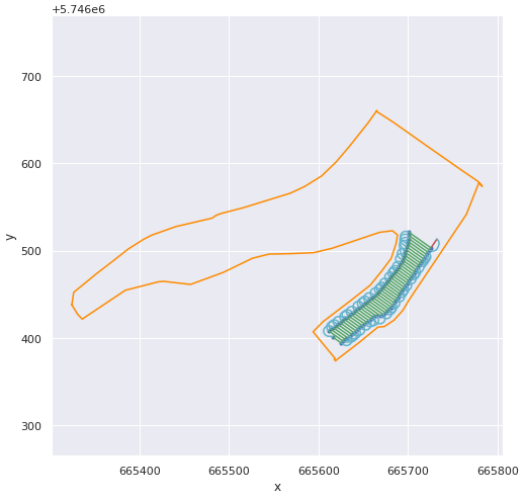
Secondly, as the field has now been divided into three smaller fields of somewhat regular shape, the Lagrangian 1-tree relaxation described in section 4.4 will be performed on these fields in order to find piecewise optimal solutions. These solutions are depicted in figure 11.



(a) 1-tree of field 3a of cost 2458.09



(b) 1-tree of field 3b of cost 530.57



(c) 1-tree of field 3c of cost 1092.13

Figure 11: 1-trees for subfields of field 3

As can be seen in figures 11b and 11c, the 1-trees found by the algorithm are Hamiltonian cycles, containing the required edges and they are therefore optimal solutions to those particular fields [9]. Thus, a piecewise optimal solution has been found for fields 3b and 3c, including their connecting edge to fields 3c and 3a respectively. However, as can be seen from figure 11a, the 1-tree obtained for field 3a is not a Hamiltonian cycle and therefore this 1-tree cannot be used in a solution route to field 3. Now, it has been attempted to divide field 3a into smaller fields and obtain an optimal solutions to these even smaller subfields of field 3a and connect these. However, the relaxation has not been able to obtain optimal solutions for these even smaller subfields either. The reason for this is to this point unknown and one can only speculate. It may simply be that this subfield is too difficult to handle for the algorithm based on its shape or one even has to conclude that the relaxation's purpose is simply only to find a tight lower bound and is not to be used to find optimal solutions.

However, as it has been possible to find optimal solutions for fields 3b and 3c, it was chosen to use the improved algorithm described in section 3.2 to find a good route on field 3a that can be connected to field 3c to eventually find a complete route for field 3. Furthermore, the idea behind this ad hoc approach remains that connecting one good route provided by the algorithm described in section 3.2 and two piecewise optimal solutions will still deliver a good result, which will come somewhat close to an optimal solution. Now, the improved algorithm has been performed on field 3a and the result is depicted in figure 12.

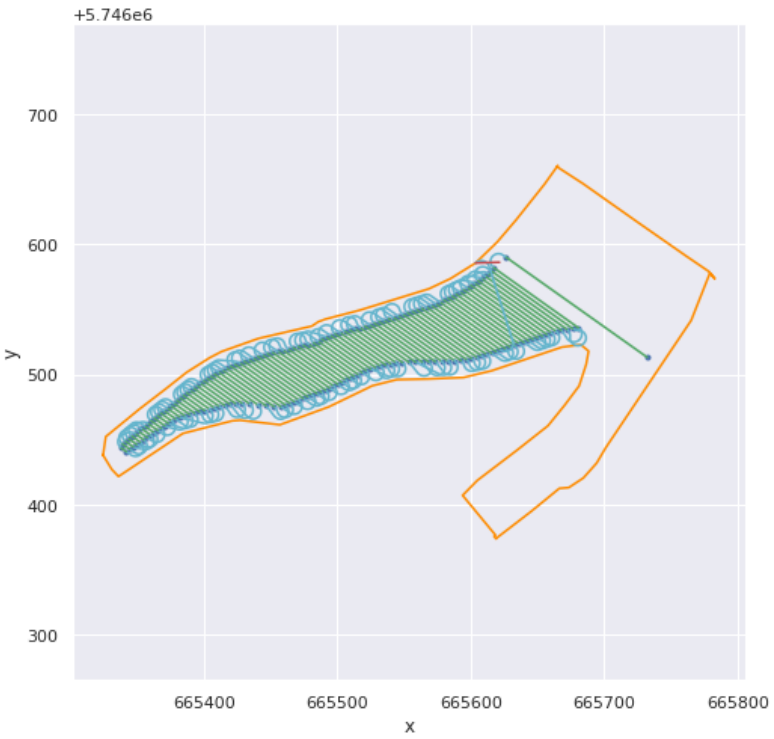


Figure 12: Route on field 3a connected to field 3c of cost 2597.20

As regards the route on field 3a obtained by the improved algorithm, it has a cost of 2597.20, in comparison to a lower bound obtained by the Lagrangian 1-tree relaxation of 2458.09. This means that the route obtained is at most 5.66% more expensive than the optimal route on field 3a and its cost can thus at best be reduced by 5.36%. Moreover, note that these results only concern field 3a and would be even less significant when taking into account the whole of field 3, as optimal routes were found for field 3b and 3c.

Now, the complete route obtained for field 3 using this ad hoc approach is depicted in figure 13. Furthermore, all results that are obtained in this thesis are presented in table 4.

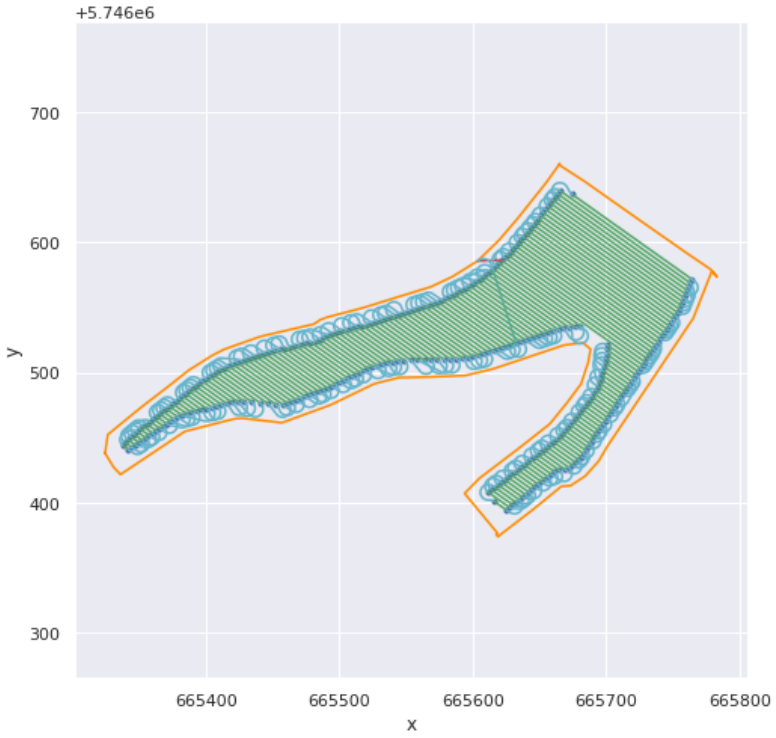


Figure 13: Ad hoc solution for field 3 of cost 4219.90

Field	Number of edges	Old algorithm	Improved algorithm	MST	1-tree	Ad hoc
1	44	1234.07	1082.06	974.45 (lb)	1059.09 (optimal)	-
2	59	1496.99	1442.12	1160.00 (lb)	1366.82 (optimal)	-
3	169	4551.39	4461.33	3807.59 (lb)	4054.00 (lb)	4219.90
3a	94	-	-	-	-	2597.20
3b	25	-	-	-	-	530.57
3c	50	-	-	-	-	1092.13

Table 4: Table containing all results obtained

Regarding the quality of the result obtained using this ad hoc approach, one can analyze the data presented in table 4. Firstly, one can immediately note that this ad hoc approach has delivered quite a significant improvement from the improved algorithm discussed in section 3.2. More specifically, the ad hoc approach has enabled one to reduce the cost of the route on field 3 with 5.41%, with respect to this algorithm. Furthermore, when comparing this result to the 1-tree obtained in section 4.6, one can also conclude that the route obtained by the ad hoc approach is at most 4.09% more expensive than an optimal solution, which, as stated in section 4.6, is probably not reachable. Thus, this means that, using this ad hoc approach on field 3, one has been able to find a route of which the cost can at best be reduced by 3.93%.

In conclusion, it has been shown that this ad hoc approach elaborated on in this section is able to provide improvements to the solution presented in section 3.2. However, as this has only been tested on 1 particular field, it remains to show that this approach generally works, yet this result is rather promising.

6 Conclusion

The goal of this thesis was to improve the current solution to the routing problem at hand. Furthermore, one also wanted to be able to assess the quality of the solutions obtained in order to determine how much room for improvement there still is and how worthwhile any time spent towards this is.

To summarize what has been done in this thesis, firstly, the problem was defined and identified as the Rural Postman Problem. From here, the notion of Dubins curves was introduced in section 2 in order to compute the turns used to connect the required edges and find routes on the agricultural fields. Then, the first solution to the problem at hand, the one currently being used, was analyzed. This is a heuristic algorithm based on Monte Carlo methods [8], which was also used in a bachelor thesis previously done on this subject for the company [15]. Some slight alterations were made to this implementation which improved the results, based on 3 test fields, quite significantly and which could be implemented by the company rather quickly.

From here, one started to tackle the second goal of this thesis, which was to obtain a lower bound in order to assess the results. Firstly, in order to obtain a simple lower bound, the notion of minimum spanning trees was introduced, computed using a slight alteration to Kruskal's algorithm, described in section 4.1. Using these minimum spanning trees a lower bound was obtained, however from the results obtained by these, one could conclude that these bounds were not very tight and needed to be improved in order to be more conclusive. Thus, it was opted to elaborate on these minimum spanning trees using a relaxation as described in section 4.2, based on the problem definition used in section 4.3. The relaxation eventually used was a Lagrangian 1-tree relaxation, described in section 4.4, and its implementation has been elaborated on in section 4.5. This implementation of the Lagrangian 1-tree relaxation yielded very interesting results, which were described and analyzed in section 4.6. Most importantly, one was able to find a verifiable optimal solution to two out of the three test fields provided and a significantly improved lower bound to the other field. Moreover, these results also enabled one to conclude that the results obtained by the improved algorithm described in section 3.2 were of good quality, yet leaving some room for improvements.

Lastly, an ad hoc approach was implemented in order to improve the results obtained for the final field to which no optimal solution was found, which has been elaborated on in section 5. Using this approach, one was able to find some significant improvements to the solution obtained by the improved algorithm, however this came with quite some difficulties which were resolved using a trial-and-error approach.

Thus, in conclusion, in this thesis, one has been able to find some significant improvements to the current solution of the routing problem at hand. These improvements can either be obtained using a slight alteration to the current solution, using the Lagrangian 1-tree relaxation which in certain cases even provides verifiable optimal solutions, or even using the ad hoc approach described. Moreover, one is now also able to assess the quality of the results obtained, using the tight lower bounds that are computed using the relaxation.

7 Discussion

In this section, one attempts to answer the question: where to go from here?

Firstly, regarding the Dubins curves, as could be seen throughout this thesis, a problem occurred from time to time where the curves went straight through the field. As has been mentioned, this is not a problem if the field has not been worked in that particular place, however it is in other instances and this is something that has to be solved manually as of right now. This is the case, as the algorithm computes the shortest possible Dubins curve, without taking into account that this might cross the field. Furthermore regarding the Dubins curves, one might want to work with continuous Dubins curves instead of the ones being used right now. Right now, the Dubins curves can either go straight ahead or take a turn along an angle that corresponds with the turning radius and there is no in between. This is not impossible, however it is not desirable for the machine's long term operating and therefore this is something that the company is actually currently working on and could be implemented in the results presented in this thesis as well. Note that this will not reduce the length of the fields, however it is a more practical application.

Secondly, something that might be worthwhile to invest some time and effort in would be the ad hoc approach described in section 5. One could see in the analysis of the results that this approach provided a rather significant improvement to the route for field 3. As mentioned before, this approach would need some more testing to conclude whether or not this is actually an approach that could consistently deliver good solutions. If this turns out to be the case, one might opt to try and automate this approach in order to consistently obtain better results. However, there are some drawbacks to this approach that have been experienced during its implementation as well. These would include the division of the field, which has now been done manually and rather intuitively. Furthermore, one might run into the same problem that occurred in field 3a, where no piecewise optimal solution could be found, even when making the field smaller and where it was therefore opted to use the algorithm described in section 3.2. Thus, one might have to include some bound at which the automation opts to use the algorithm from section 3.2 instead of trying to find a piecewise optimal solution using the Lagrangian 1-tree relaxation.

References

[1] Overview of NetworkX - NetworkX 2.4 documentation, 2019 (accessed June 26, 2020). `https://networkx.github.io/documentation/stable/`.

[2] AgXeed, 2020 (accessed June 29, 2020). `https://agxeed.com/`.

[3] Phact - Innovation Experts, 2020 (accessed June 29, 2020). `https://phact.nl/`.

[4] Egon Balas and Paolo Toth. Branch and Bound Methods for the Traveling Salesman Problem. *December, 1983, 65 p. : ill., tables. Includes bibliography*, 1983.

[5] Jean-Daniel Boissonnat, André Cerezo, and Juliette Leblond. Shortest path of bounded curvature in the plane. Technical report, 1994.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. Technical report, 2009.

[7] L. E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497, 7 1957.

[8] P. Fernández De Córdoba, L. M. García Raffi, and J M Sanchis. A heuristic algorithm based on monte carlo methods for the rural postman problem. *Computers and Operations Research*, 25(12):1097–1106, 1998.

[9] A. M. Geoffrion. Duality in Nonlinear Programming: A Simplified Applications-Oriented Development. *SIAM Review*, 13(1):1–37, 1 1971.

[10] Gregory Gutin and Abraham Punnen. The traveling salesman problem. *Discrete Optimization*, 3(1):1, 2006.

[11] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 12 1971.

[12] Harold H. Johnson. An application of the maximum principle to the geometry of plane curves. *Proceedings of the American Mathematical Society*, 44(2):432–432, 2 1974.

[13] S.M. Lavalle. 15.3.1 Dubins Curves. *Cambridge University Press*, pages 2–6, 2006.

[14] Nicolaos Matsakis. Solving the Rural Postman problem using the Adleman-Lipton model. *arXiv*, pages 1–8, 12 2010.

[15] Laura Philipse. *Routing Algorithms for Autonomous Agricultural Vehicles*. PhD thesis, Radboud University, 2020.

[16] BT Poljak. Subgradient methods: A survey of Soviet research. *IIASA PROCEEDINGS SERIES*, pages 1–21, 1977.

[17] Andrei M Shkel and Vladimir Lumelsky. Classification of the Dubins set. *Robotics and Autonomous Systems*, 34(4):179–202, 2001.

[18] Ton Volgenant and Roy Jonker. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research*, 9(1):83–89, 1 1982.

[19] Andrew Walker. dubins · PyPI, 2018 (accessed July 11, 2020). `https://pypi.org/project/dubins/`.

A Appendix

A.1 Code for heuristic algorithm

```
!pip install dubins
import matplotlib.pyplot as plt
import math
import numpy as np
import itertools
import random
import timeit
import dubins
from copy import copy, deepcopy

def readArray(lines, field):
  """ Translates the lines and fields into two separate lists of x and y
  coordinates
  """
  x = []
  y = []
  a = []
  b = []
  for i in lines:
    x.append(i[0][0])
    x.append(i[1][0])
    y.append(i[0][1])
    y.append(i[1][1])
  for i in field:
    a.append(i[0])
    b.append(i[1])
  return x, y, a, b

def calcArc(dx,dy):
  """return argument of dx + i dy
  -pi/2 <= arg < 3/2pi"""
  ai = 0.0
  if dx == 0:
    if dy > 0:
      ai = math.pi/2
    elif dy < 0:
      ai = -math.pi/2
  elif dx > 0:
    ai = math.atan(dy/dx)
  else:
    ai = math.atan(dy/dx) + math.pi
  return ai

def calculateTurnsNew(xlines, ylines, turnRad):
  """ Calculates the Dubins curves
  """
  turns = []
  for i in range(0,len(xlines)):
    m = []
    if i%2 == 0:
```

```
      ii = i+1
    else:
      ii = i-1
    dxi = xlines[i] - xlines[ii]
    dyi = ylines[i] - ylines[ii]
    ai  = calcArc(dxi,dyi)
    q1  = (xlines[i], ylines[i], ai)
    # A configuration is (x, y, theta), where theta is in radians,
    # with zero along positive x-axis,
    # and taking counter-clockwise as positive
    for j in range(0,len(xlines)):
      if j%2 == 0:
        jj = j+1
      else:
        jj = j-1
      dxj = xlines[jj] - xlines[j]
      dyj = ylines[jj] - ylines[j]
      aj  = calcArc(dxj,dyj)
      q2  = (xlines[j], ylines[j], aj)
      dub = dubins.shortest_path(q1,q2,turnRad)
      m.append(dub)

    turns.append(m)
  return turns

def calculateDistanceMatrix (xlines, ylines, turns):
  """ Calculates the distance matrix using the dubins curves
  """
  distance = []
  for i in range (0,len(xlines)):
    m = []
    for j in range (0,len(xlines)):
      if i==j:
        m.append(0)
      elif ((i%2==0 and j%2==0) or (i%2==1 and j%2==1)): #turn
        val = turns[i][j].path_length()
        m.append(val)

      elif ((i%2==0 and j-i==1) or (j%2==0 and i-j==1)): #required edge
        val = 0 #as it is fixed
        m.append(val)
      else:
        m.append(math.inf)
    distance.append(m)
  return distance

def makeBasicRoute(length):
  """ Makes a naive route
  """
  z = list(range(0, length))
  for i in range(0, length):
    for j in range(0, length):
      if (i%4==2 and j%4==3 and j-i == 1):
        swap = z[i]
```

```
            z[i] = z[j]
            z[j] = swap
    return z

def plotRoute(route, xlines, ylines, xfield, yfield, turns):
    """ Plots the route computed
    """
    fig, ax = plt.subplots( )
    ax.plot(xfield, yfield)

    xx = []
    yy = []
    for i in range (0, len(route)):
      xx.append(xlines[route[i]])
      yy.append(ylines[route[i]])
      if (i+1<len(route) and ((route[i]%2==0 and route[i+1]%2==0)
      or (route[i]%2==1 and route[i+1]%2==1))):
        con = turns[route[i]][route[i+1]].sample_many(0.1)
              #sample rate doesn't change distance
        a = [q[0] for q in con[0]]
        b = [q[1] for q in con[0]]
        for i in a:
          xx.append(i)
        for i in b:
          yy.append(i)

    line, = ax.plot(xx, yy)
    line2, = ax.plot(xlines, ylines, '.')
    line3, = ax.plot(xfield, yfield)
    line4, = ax.plot(xfield[0:2], yfield[0:2], color = 'blue')

    ax.set_xlabel("x")
    ax.set_ylabel("y")
    plt.show()

def checkRoute(route):
    """ Checks whether or not a computed route is actually a route by
    checking if all required nodes are in it
    """
    numPairs = len(route)/2
    for i in range(0, len(route), 2):
      if (abs(route[i+1] - route[i])==1):
        numPairs -= 1
    if (numPairs == 0):
      return True
    return False

def solutionLength (solution, distance, stop):
    """ Calculates the length of a computed route, this does not include
    the edge from the main land to the field
    """
    length = 0
    for i in range(1, len(solution)):
      if (length>=stop):
```

```
      break
    length = length + distance[solution[i-1]][solution[i]]
  return length

def fieldToMain(xlines, ylines, field, k, turningRadius):
  """ Calculates the dubins curve and length from a given point on the field
  to any point on the main land, which will be the last edge of the route
  """
  length = []
  arc0 = calcArc(field[k+1][0] - field[k][0], field[k+1][1] - field[k][1])
  q0 = (field[k][0], field[k][1], arc0)
  for i in range(len(xlines)):
    if i%2 == 0:
      ii = i+1
    else:
      ii = i-1
    dxi = xlines[ii] - xlines[i]
    dyi = ylines[ii] - ylines[i]
    arc1  = calcArc(dxi,dyi)
    q1 = (xlines[i], ylines[i], arc1)
    d = dubins.shortest_path(q0, q1, turningRadius)
    dist = d.path_length()
    length.append(dist)
  return length

def paper41(length, distance, alpha, iterations, x, distmain):
  """ Implementation of the heuristic algorithm based on monte carlo methods
  """
  shortestRoute = []
  shortestDist = math.inf
  for i in range(0,iterations):
    route = []
    route.append(0)
    for j in range (1,length):
      if (route[j-1]%2==0 and route[j-1]+1 not in route):
        route.append(route[j-1]+1)
      elif (route[j-1]%2==1 and route[j-1]-1 not in route):
        route.append(route[j-1]-1)
      else:
        options = []
        for q in range (0, len(distance[j-1])):
          if (distance[route[j-1]][q]!=math.inf) and (q not in route):
            options.append(q)
        if (options):
          opt = options
          total = 0
          p = []
          for k in opt:
            a =1/(distance[route[j-1]][k])**alpha
            p.append(a)
            total = total + a
          for k in range(0, len(opt)):
            add = p[k]*(1/total)*x
            for g in range(0,int(add)-1):
```

```
            options.append(opt[k])

          ran = random.randint(0,len(options)-1)
          route.append(options[ran])
        else:
          print("no options")
          break

    if (len(route)==length):
      if checkRoute(route):
        distfieldmain = distmain[route[length-1]]
        dist = solutionLength(route, distance, shortestDist) + distfieldmain
        # edge from field to main land
        if (dist < shortestDist):
          shortestDist = dist
          shortestRoute = route
  return shortestRoute, shortestDist
```

A.2 Code for Lagrangian 1-tree relaxation

```
!pip install dubins
import math
import networkx as nx
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
import itertools
import random
import timeit
import dubins
from typing import Dict, Tuple

def calcArc(dx,dy):
    """return argument of dx + i dy
    -pi/2 <= arg < 3/2pi"""
    ai = 0.0
    if dx == 0:
        if dy > 0:
            ai = math.pi/2
        elif dy < 0:
            ai = -math.pi/2
    elif dx > 0:
        ai = math.atan(dy/dx)
    else:
        ai = math.atan(dy/dx) + math.pi
    return ai

def calcEdgeWeight( nd1, mt1, nd2, mt2, turnRad):
    """ Calculates the weight of an edge
    """
    if nd1 == nd2:
        return math.inf
    # return into same track not allowed
    dx1 = nd1[0] - mt1[0]
    dy1 = nd1[1] - mt1[1]
    a1 = calcArc(dx1,dy1)
    dx2 = -nd2[0] + mt2[0]
    dy2 = -nd2[1] + mt2[1]
    a2 = calcArc(dx2,dy2)
    q1 = (nd1[0], nd1[1], a1)
    q2 = (nd2[0], nd2[1], a2)
    dub = dubins.shortest_path(q1,q2,turnRad)
    return dub.path_length()

def plotGraph(G,lines,field,startnode,endnode,startarc,turnRad):
    """ Plots a graph based on input such as a given graph,
    a start and end node and startarc
    """
    #fig, ax = plt.subplots( )
    #line1, = ax.plot(xlines, ylines, '.')
```

```
#line2, = ax.plot(xfield, yfield)

xlines,ylines,xfield,yfield = readArrays(lines,field)
fig, ax = plt.subplots( figsize = (8,8))
line1, = ax.plot(xlines, ylines, '.',color='b')
line2, = ax.plot(xfield, yfield,color='darkorange')

maxx,minx = max(xfield), min(xfield)
maxy,miny = max(yfield), min(yfield)
width = max(maxx-minx,maxy-miny)
midx,midy = (maxx+minx)/2,(maxy+miny)/2
leftx,rightx = midx-0.55*width,midx+0.55*width
lefty,righty = midy-0.55*width,midy+0.55*width
ax.set_xlim(leftx,rightx)
ax.set_ylim(lefty,righty)

ax.set_xlabel("x")
ax.set_ylabel("y")

mate = nx.get_node_attributes(G,'mate')
ge = G.edges()
for e in ge:
    (p,q) = e
    xe = [ p[0],q[0] ]
    ye = [ p[1],q[1] ]
    if q in mate.keys() and p == mate[q]:
        ax.plot(xe,ye,color='g')
    elif startnode in e and endnode in e:
        ax.plot(xe,ye,color='r')
    else:
        if startnode in e:
            q1 = (startnode[0],startnode[1],startarc)
            if startnode == e[0]:
                nd2 = e[1]
            else:
                nd2 = e[0]
            mt2 = mate[nd2]
            dx2,dy2 = mt2[0]-nd2[0],mt2[1]-nd2[1]
            a2 = calcArc(dx2,dy2)
            q2 = (nd2[0], nd2[1], a2)
        else:
            (nd1,nd2) = e
            mt1 = mate[nd1]
            dx1,dy1 = -mt1[0]+nd1[0],-mt1[1]+nd1[1]
            a1 = calcArc(dx1,dy1)
            q1 = (nd1[0], nd1[1], a1)
            mt2 = mate[nd2]
            dx2,dy2 = mt2[0]-nd2[0],mt2[1]-nd2[1]
            a2 = calcArc(dx2,dy2)
            q2 = (nd2[0], nd2[1], a2)
        turn = dubins.shortest_path(q1,q2,turnRad)

        con = turn.sample_many(0.1) #sample rate doesn't change distance
        a = [q[0] for q in con[0]]
```

```
            b = [q[1] for q in con[0]]
            ax.plot(a, b, color='c')

    plt.show()

def buildgraph(lines,turnRad) -> nx.classes.graph.Graph:
    """builds a complete, weighted graph on line nodes"""
    nodeset = []
    mateset = []
    for pr in lines:
        nodeset.extend([tuple(pr[0]),tuple(pr[1])])
        mateset.extend([tuple(pr[1]),tuple(pr[0])])
    mate = {}
    for mt in zip(nodeset,mateset):
        mate[mt[0]] = mt[1]
        mate[mt[1]] = mt[0]
    G = nx.complete_graph(nodeset)
    v = [( n, {'mate': mate[n]} ) for n in nodeset ]
    G.add_nodes_from(v)
    e = []
    for nd1 in nodeset:
        for nd2 in nodeset:
            wght = calcEdgeWeight( nd1, mate[nd1], nd2, mate[nd2], turnRad)
            e.append((nd1,nd2,wght))

    G.add_weighted_edges_from(e)
    e2 = []
    for nd in nodeset:
        mt = mate[nd]
        e2.append((nd,mt,0.0))
    G.add_weighted_edges_from(e2)
    return G

def createStartEnd(G,field, lines):
    """picks start/end node from field description, define start direction,
    and final node in graph G"""
    startNode = tuple(field[0])
    startDir = (field[1][0] - field[0][0], field[1][1] - field[0][1])

    mindist2 = math.inf
    endNode = startNode
    for n in list(G.nodes()):
        sqrdist = (startNode[0]-n[0])**2 + (startNode[1]-n[1])**2
        if mindist2 > sqrdist:
            endNode = n
            mindist2 = sqrdist
    startArc = calcArc(startDir[0],startDir[1])

    return startNode, endNode, startArc

def buildstar(G,startNode,endNode,startArc) -> nx.classes.graph.Graph:
    """defines edges, with weights, linking start and endnode to graph;
    edge between start and end node has weight zero"""
```

```
    starGraph = nx.Graph()
    ndlist = [ startNode ]
    ndlist.extend(list(G.nodes()))
    nx.add_star(starGraph, ndlist)
    mate = nx.get_node_attributes(G,'mate')
    q1 = ( startNode[0], startNode[1], startArc )
    enw = list(starGraph.edges())
    ew = []
    for ee in enw:
        nd2 = ee[1]
        mt2 = mate[nd2]
        dx,dy = mt2[0]-nd2[0], mt2[1]-nd2[1]
        arc2 = calcArc(dx,dy)
        q2 = ( nd2[0], nd2[1], arc2 )
        wght = dubins.shortest_path(q1,q2,turning_radius).path_length()
        if nd2==endNode:
            ew.append((startNode,endNode,0.0))
        else:
            ew.append((startNode,nd2,wght))
    starGraph.add_weighted_edges_from(ew)

    return starGraph

def buildFullGraph(lines,field,turnRad) -> (nx.classes.graph.Graph,Tuple,Tuple):
    """builds instance in form of weighted graph,
    returns weighted graph, start and end of route to be built
    nodes u,v forming a line are each others mate"""

    testG = buildgraph(lines,turnRad)
    startNode,endNode,startArc = createStartEnd(testG,field, lines)
    endMate = nx.get_node_attributes(testG,'mate')[endNode]
    endNbrs = [ n for n in testG[endNode] if n != endMate ]
    endE = [ (endNode, nd, {'weigth': math.inf}) for nd in endNbrs ]
    testG.add_edges_from(endE)
    starGraph = buildstar(testG,startNode,endNode,startArc)
    fullgraph = nx.Graph(testG)
    fullgraph.add_edges_from(list(starGraph.edges(data=True)))

    return fullgraph, startNode, endNode, startArc

def oneTree(G,labda : Dict) -> (float,nx.classes.graph.Graph):
    """compute 1-tree relaxation of TSP in weighted graph G, with
    edge weight adjusted by
    degree-2 multipliers labda, we do not a priori fix special node"""
    newG = nx.Graph(G) #make a copy
    curW = nx.get_edge_attributes(newG,'weight')
    newlabda = {}  #to prevent key errors
    for v in list(newG.nodes):
        newlabda[v] = labda.get(v,0.0)
    newW = []  #to modify existing weights
    for e in newG.edges():
        nw = curW[e] + newlabda[e[0]] + newlabda[e[1]]
        newW.append((e[0], e[1], {'weight': nw}))
    newG.add_edges_from(newW)
```

```
    mst = nx.minimum_spanning_tree(newG)
    ew = sum([ e[2]['weight'] for e in mst.edges(data=True) ])
    leafset = set([ vd[0] for vd in list(mst.degree()) if vd[1]==1 ])
    worstEdge = (-math.inf,None,None) # holding current worst edge to be added
    for lf in leafset:
        lfnbr = [n for n in mst[lf]][0] #neighbor in mst
        grnbr = [n for n in newG[lf]]   #neighbors in graph
        #look for min cost neighbor other than lfnbr
        minnd, minw = None, math.inf
        for v in grnbr:
            vw = newG[lf][v]['weight']
            if v != lfnbr and vw < minw:
                minnd, minw = v, vw
        if minw > worstEdge[0]:
            worstEdge = (minw,minnd,lf)

    xw, nbr, lf = worstEdge
    ew += xw
    mst.add_edges_from([(lf, nbr, {'weight' : xw})])

    return ew, mst

def seqOneTree(G : nx.classes.graph.Graph,
               maxIter : int, maxFail : int,
               printFlag = False,
               initStep = 5.0, decFac = 0.99) -> (float,Dict):
    """compute sequence of oneTree relaxations for weighted graph G,
    terminating after maxIter iterations
    or after maxFail consecutive iterations without improvement.
    Return highest lower bound encountered, with corresponding lambda's """
    curLab = dict([ (v,0.0) for v in list(G.nodes()) ])
    wght,otree = oneTree(G,curLab)
    bestW = wght
    bestLab = dict([ (v,0.0) for v in list(G.nodes()) ])
    degdfPred = dict([ (v,0) for v in list(G.nodes()) ])
    degdfCur = dict([ (vd[0],vd[1]-2) for vd in list(otree.degree()) ])

    itcnt = 1
    if printFlag:
        print("use initStep = "+str(initStep)+", decrement factor: "+str(decFac))
        print("it 1 : wght: "+str(wght))
    successiveFailures = 0
    curStep = initStep
    while itcnt < maxIter and successiveFailures < maxFail:
        itcnt += 1
        newLab = dict([ (v,curLab[v]+curStep*(0.7*degdfCur[v]+0.3*degdfPred[v]))
                        for v in list(G.nodes()) ])
        nwght, ntree = oneTree(G,newLab)
        # check for improvement
        if nwght > bestW:
            bestW = nwght
            bestLab = newLab
            successiveFailures = 0
            if printFlag:
```

```
                print("it "+str(itcnt)+" : wght: "+str(nwght))
        else:
            successiveFailures += 1
        # update for next iteration
        curStep *= decFac
        curLab = newLab
        degdfPred = degdfCur
        degdfCur = dict([ (vd[0],vd[1]-2) for vd in list(ntree.degree()) ])

    if printFlag:
        print("quitting after "+str(itcnt)+" iterations")

    return bestW, bestLab
```