Anonymized summary of master thesis

Developing a routing algorithm and a prediction method for the turtle rescue problem

Author: Jan Groeneveld

University of Twente

Supervisor: Dr. Ir. J.M.J. Schutten

Supervisor: Dr. Ir. M.R.K. Mes

Graduation date: 18.10.2017

Table of Contents

1 Introduction

This project is part of the Master program Industrial Engineering and Management at the University of Twente. Due to confidentiality of the conducted research, only a summary is available until 1.6.2019. This summary is anonymized and contains only the most valuable parts of the conducted research regarding its contribution to the literature. In this chapter, we introduce the problem of this research by first defining it in Section 1.1 and then explaining our plan of approach in Section 1.2.

1.1 Problem definition

In our research, we deal with a new generalization of the *team orienteering problem* (TOP). Chao, Golden, and Wasil (1996) define the TOP as the problem where a team of competitors starts at the same point, visits different locations in order to collect a certain reward, and finally returns to the starting point within a limited amount of time. It is the objective to maximize the sum of rewards of all competitors, whereby a node may be visited only once a day. The time a competitor travels between two nodes is denoted as travel time and the time that the competitor stays at a node is denoted as service time.

In the literature the TOP is also associated with other terms such as the multiple tour maximum collection problem (MTMCP). However, Feillet et al. (2005) state that the TOP differs from the MTMCP as the TOP is generally defined as paths rather than circuits. However, by "adding a dummy arc from the destination to the origin of the paths makes the two problems equivalent" (Feillet et al., 2005, p.189). In the literature, we find different applications:

- Scheduling maintenance technicians problem (Tang, Miller-Hooks & Tomastik, 2007)
- Tourist route planning problem (Gavalas et al., 2015; Vansteenwegen et al., 2009a; Vansteenwegen et al., 2009b; Vansteenwegen et al., 2009c)
- Bank robber problem (Awerbuch, Azar, Blum & Vempala, 1998)
- Home fuel delivery problem (Tang & Hooks, 2005)
- Athlete recruiting problem (Tang & Hooks, 2005)

In recent literature, different generalizations of the TOP are discussed, such as the TOP with time windows, the TOP with time dependent and/or stochastic travel times, service times, and rewards (Verbeeck et al., 2014b). However, in this research, we present a new generalization, which is too our knowledge not yet introduced to the literature. In this generalization, the competitors are allowed to visit a location multiple times a day. The rewards at the nodes are inter-related because the reward of a visit depends on the time difference to an earlier visits if the node has already been visited the same day. In addition, we consider time-dependent service times, travel times, and rewards. Therefore, we call this generalization the time-dependent TOP with multiple visits (or split deliveries) and inter-dependent rewards, which we denote as TD-TOPMV. This problem could have different applications, such as various inspection, collection, or salesmen problems.

In this summary, we specifically describe the "turtle rescue problem" as an example of the TD-TOPMV. In this case, however, the problem also has a periodical planning horizon and multi constraints due to different stopping points, stopping times, and shift changes. Therefore, we call it the TD-PTOPMVMC. In this problem, there is an organization that wants to maximize the number of saved sea turtles. Every day, a fleet of homogeneous competitors leaves from one depot to go to different beaches. At every beach, they can find a certain number of turtles. A beach can be visited multiple times on one day. Every beach belongs to a certain region. Next to maximizing the number of saved turtles, two additional objectives have to be taken into account, whereby the first one is more important:

1. Save a certain number of turtles in every region in every month.
2. Visit every beach every three days.

1.2 Plan of approach

This section describes the plan of approach of this research, which also includes the research questions. Before introducing all research questions, we present our research goal:

"Develop a prediction method and a routing algorithm that maximizes the number of saved turtles and takes the additional objectives into account"

From this research goal, we derive research questions, which are discussed in the following chapters:

Chapter 2 – Literature

This chapter introduces the required literature of this thesis. First of all, we want to know how similar routing problems and especially the TOP have been tackled and solved. Furthermore, we develop a method that predicts the number of saved turtles at a beach at a certain time. Consequently, we answer the following questions:

- What is known in the literature about the complexity of the proposed problem?
- What is known in the literature about algorithms to solve the original TOP?
- What is known in the literature about prediction models?

For the purpose of this literature research, we use Scopus and Google Scholar.

Chapter 4 – Prediction method

Within this chapter, we propose a possible prediction method. Furthermore, we discuss the choices with regards to the algorithm and the strategies behind it.

- What kind of prediction method is most suitable?

Chapter 5 - Routing algorithm

Within this chapter, we design our routing algorithm. Furthermore, we discuss the choices with regards to the algorithm and the strategies behind it.

- What kind of algorithm is most suitable for this problem?
- How can we measure the performance of the algorithm?

Chapter 6 – Conclusion

This chapter provides the conclusion of our research.

2 Literature

In our summary of the literature study, we focus on the complexity of the proposed problem, possible heuristics to solve the TOP, and present different prediction models.

2.1 Complexity

The single variant of the TOP, the orienteering problem (OP), is defined as a combination of the knapsack problem, which maximizes an objective function by choosing items subject to a packing constraint (Hochbaum, 1995), and the well-known traveling salesperson problem (Verbeeck, Sörensen & Aghezzaf, 2014a). That is because in order to solve this problem not only the determination of the route is needed but also the subset of the nodes that will be visited have to be chosen (Verbeek, Aghezzaf & Vansteenwegen, 2014). Our problem including the possibility of multiple visits a day, extends the problem because it requires that also the number of times a node is visited has to be determined. Regarding the running time complexity, the TSP is known to be NP-hard. Since the TOP includes an added element of complexity, it follows that the TOP is also NP-hard (Butt & Ryan, 1999). With another added element of complexity, the same holds logically for the TOPMV (and the TD-PTOPMVMC). In our research, we concluded that an exact algorithm cannot solve this problem within reasonable time and therefore we only considered heuristics to solve this problem.

2.2 Team orienteering problem heuristics

A lot of different heuristics have been introduced in literature to solve different variations of OP, STSP, MTMCP, TOP, and MTMCP. In this section, we describe the most important ones. According to Vansteenwegen et al. (2011), the best-performing TOP algorithms are discussed in Tang and Miller-Hooks (2005), Archetti et al. (2007), Ke et al. (2008), Vansteenwegen et al. (2009c), and Souffriau et al. (2010). The computational results of these algorithms are shown in Table 1.

Reference	Computer specifications	Technique	Algorithm	# best	Avg gap (%)	Avg CPU (seconds)
Tang and Miller-Hooks (2005)	DEC Alpha XP1000, 1 GB RAM, 1.5 GB swap	Tabu search	TMH	34	1.32	336.6
Archetti et al. (2007)	Intel Pentium 4, 1 GB RAM, 2.8 GHz	Tabu search	TSF	94	0.20	531.5
			TSU	69	0.49	318.0
		Variable neighbourhood search	SVN	**128**	**0.05**	906.1
			FVN	97	0.18	**63.6**
Ke et al. (2008)	PC, 3.0 GHz	Ant colony optimisation	ASe	**130**	**0.08**	252.3
			ARC	81	0.40	204.8
			ADC	80	0.35	213.8
			ASi	84	0.32	215.0
Vansteenwegen et al. (2009c)	Intel Pentium 4, 1 GB RAM, 2.8 GHz	Variable neighbourhood search	SVNS	44	0.97	**3.8**
Souffriau et al. (in press)	Intel Xeon, 4 GB RAM, 2.5 GHz	Greedy randomised adaptive search procedure with path relinking	FPR	78	0.39	**5.0**
			SPR	**131**	**0.04**	212.4

Table 1 – Summary of the best-performing TOP algorithms (Vansteenwegen et al., 2011, p.5)

Concerning these articles, Vansteenwegen (2009b) argues that the local search moves used in these TOP solutions are not effective when applied to time windows because they include local search moves that become useless when time windows are considered. Time windows are not that big a problem but there are different reasons that make local search moves difficult to implement. First of all, the travel time, service time, and expected number of saves turtles are time-dependent and that means that by applying local search techniques, such as swapping, we need to calculate the entire route of that competitor again and have to take into account that no time restrictions are violated. Moreover, as explained in Section 1.2, a beach can be visited multiple times a day and when that happens the rewards are inter-related. This leads to a recomputation of the rewards of the entire day planning. Therefore, local search moves will be difficult to implement but we do consider these algorithms as well

(except for the article of Archetti et al. (2007) because is not accessible for us) to get a broader impression of possible solution heuristics. Finally, we consider these algorithms and also some that are applied to the TOP with time windows.

2.2.1 Tabu search embed in an adaptive memory procedure

Tang & Miller-Hooks (2005) applied a tabu search heuristic embedded in an adaptive memory procedure. However, instead of reviewing this article, we review the article from Tang, Miller-Hooks & Tomastik (2007) because they use the same approach and extend the TOP by considering time-dependency and a periodical planning, which fits better to our problem.

Tang et al. (2007) tackle the problem of scheduling technicians for planned maintenance. They consider a planning period of 3 weeks and time-dependent rewards to better describe the reality. Greater rewards are assigned to locations that have not been maintained for a longer time. The travel times between locations and service times at every location are different but not time-dependent. Their approach includes three AMP steps:

- Partial solution generation and storage:

  Partial solutions are defined as one single tour of the m tours. First, a set of partial solutions is generated and stored. The first non-depot vertex is randomly chosen. Random vertices are added in between a pair of vertices, which depends on a ratio with regards to the added tour duration and the added reward.

- Solutions construction:

  Afterwards, solutions are constructed by combining partial solutions. The selection preference is biased to those single tours with preferred objective values. All constructed solutions are improved by tabu search afterwards. Both random and greedy procedures are applied in the neighborhood solution exploration.

- Partial solution update:

  The solutions maintained in the adaptive memory are updated with these improvements. Low-reward tours in the adaptive memory are replaced by the improved tours.

2.2.2 Ant colonization optimization

Montemanni and Gambardella (2009) apply an ACO heuristic to the team orienteering problem with time windows. They define their problem as a hierarchical TOP, which requires the same input as the TOP does but it requires a set of non-overlapping elementary paths, which have an ordered sequence of nodes starting from node 1 and ending at node n.

The construction phase is performed by sending out all ants sequentially. Iteratively, every ant goes probabilistically from node i to node j based on the *pheromone trail* and the *desirability*. The pheromone trails contain the trails of previous ants that travelled there and indicate how good this path has been in the past. The desirability is a formula regarding the associated profit, the distance, and the time window of node j. The possible nodes for j are selected out of a set of feasible nodes, which still need to be visited and are within the time window. Note that only the best ant, which collected the most rewards, is allowed to leave a trail that is updated to all arcs. While the ant builds the solution, the pheromone trail is updated as well. Each ant removes pheromone trails of the visited arcs to make sure that there is a variety of generated solutions. Afterwards, the constructed solutions are being optimized by a local search algorithm. They apply a CROSS exchange procedure that exchanges two sub-chains of customers of the giant tour.

Ke et al. (2007) also apply ACO but to the regular TOP without any time-dependencies or time windows. They state that sending the ants sequentially results in the best results. Furthermore, they performed a benchmark of their algorithm with the one of the Archetti and Tang et al. (2005) with the result that the quality of their solution could compete with the others but with a much faster computational time. The

results can be seen in Appendix B. Another interesting aspect of their approach is that in their heuristic function they include the angle at beach i between the way to the depot n and the next beach j. By doing so, the algorithm can send the competitor in the desired direction. First leaving the depot and then forcing the competitor more towards the depot.

Verbeeck et al. (2014a) apply ACO to a TOP with time-dependent travel times. They speed up the time-dependent insertion procedure by using a local evaluation metric. Verbeeck et al. (2014b) tackle the TOP with time-windows and time-dependent and stochastic rewards and time-dependent travel times by using a greedy randomized adaptive search procedure and a stochastic version of the ACO.

2.2.2.1 Simulated annealing

Lin and Yu (2015) apply an SA heuristic, which we briefly introduced in Section 3.2.1., for the multi-constraint TOP with multiple time windows. Their heuristic starts by creating a random initial solution. Afterwards, the initial solution is optimized by means of SA including a swap, insertion, or inversion procedure in every iteration. Additionally, they add a restart strategy as an extra diversification to avoid local optima. They state that sometimes accepting wore solutions is not enough to escape the local optima. The current temperature, which determines the probability of accepting worse solutions, decreases after every iteration. The algorithm restarts if the current best solution has not improved for a pre-determined number of consecutive temperature decreases. Once the algorithm restarts, the current temperature is reset to the initial temperature and a new initial solution is generated randomly to initiate a new SA run. They show that SA with a restart strategy is a promising heuristic method to solve multi-constraint TOP with multiple time windows and that the restart strategy enhances the performance of the SA.

2.2.2.2 Variable beach search

Tricoire, Romauch, Doerner and Hartl (2010) deal with a multi-periodic TOP with multiple time windows and use a VNS. Before applying the VNS, they first construct solutions. To this end, they use the best insertion heuristic. The insertion heuristic is based on two criteria. One is the lowest increase in distance and the other one the lowest increase in time. The feasibility of the insertions is checked by means of an exact feasibility algorithm, which operates in polynomial time. Afterwards, VNS is applied to improve the initial solution. A stopping condition can be a limit on computational time, the number of iterations, or the number of iterations without improvement. They apply the number of iterations as a stopping criterion. For every iteration of the VNS algorithm, an improvement method that depends on the number of iterations, that have been performed already (iteration 1-8: cross-exchange, iteration 9-12: optional exchange, iteration 13-17: optional exchange), uses random nodes to create a new solution. If the new solution is better, it replaces the initial solution. In a benchmark, they show that their VNS algorithm is a viable option for all kind of orienteering problems, with or without time windows.

2.2.2.3 Path relinking heuristic with a greedy randomized adaptive search procedure

Souffria et al. (2010) use a path relinking metaheuristic in combination with a greedy randomized adaptive search procedure because path relinking heuristics have been proved to work well on knapsack problems. Their approach works as follows:

While the number of iterations without improvement is not exceeded:

Construct: The construction heuristic is based on a greedy randomized adaptive search procedure. This procedure depends on a “greediness” parameter that lies between 0 and 1. This parameter indicates the level between randomness (0) and greediness (1). The parameter is determined randomly before the construction.

Local search: The local search algorithm uses 2-Opt, swap, replace and insert procedures until a local optimum is reached.

Link to elites: This procedure combines the solution, that was constructed and improved in the prior two phases, with one of the solutions out of the elite pool. The two solutions are first combined, then adapted, and finally improved to create a new feasible solution. This procedure is done for all possible combinations, therefore for all members of the pool of elites.

Update elite pool: The best solution found in the prior step is considered for the insertion into the pool of elite solutions. If the pool is full, it replaces the worst elite solution if it leads to an improvement. Every solution is assigned to an age and it increases with every time the "Link to elites" is performed. At a certain age, the solution is deleted from the pool.

2.2.2.4 Iterated local search heuristic

Vansteenwegen et al. (2009b) apply an iterated local search heuristic algorithm to the TOPTW with the purpose of developing an electric tour guide. The electric tour guide required a short computation time and therefore they chose an algorithm that is very simple, fast, and effective. They achieved this goal with an average performance gap of 1.8% to the best-known solutions and the average computation time is more than a 100 times faster than the best-known solutions. Gavalas et al. (2014, p.19) state that it is "the fastest known algorithm proposed for the TOPTW". Their approach includes an insert step in combination with a shaking step to escape from local optima that perform performs very well on a large and diverse set of the instance.

The insertion step adds one by one new visits to a tour. Before a new visit can be added, the time windows need to be checked for feasibility. A feasible node with the cheapest insertion time will be inserted. For each node, a ratio is calculated that incorporates the profit and the delay of adding this node. Afterwards, a shake step is used to escape from local optima. In this shake step, random node(s) are removed in every tour to make space for nodes that might improve the solution.

2.3 Prediction models

A common way to develop prediction models for quantitative outputs are regression models (Larsen & Marx, 2012). Regressions models analyze the effect of the independent variables, also called predictors or features, on the outcome variable. A regression analysis can include one (simple regression) or more predictors (multiple regression). The relationship between predictors and the outcome variable can be linear, curvilinear, or nonlinear. For the prediction of probabilities, logistic regressions can be applied. For numerical outputs usually a (multiple) linear regression or nonlinear regression is used.

There are also machine learning techniques that are using regression models such as artificial neural networks. These networks are inspired by the architecture of biological neural networks (Mair et al., 2000). Every network consists of neurons which are interconnected by strings. A neuron receives an input which is associated with a weight. If the sum of these weighted inputs exceed a certain threshold, the neuron fires and creates a positive or negative output for other neurons in the network. This process stops when one or more outputs are generated. An example of this process is shown in Figure 7. This example shows n inputs. If the threshold is exceeded, the

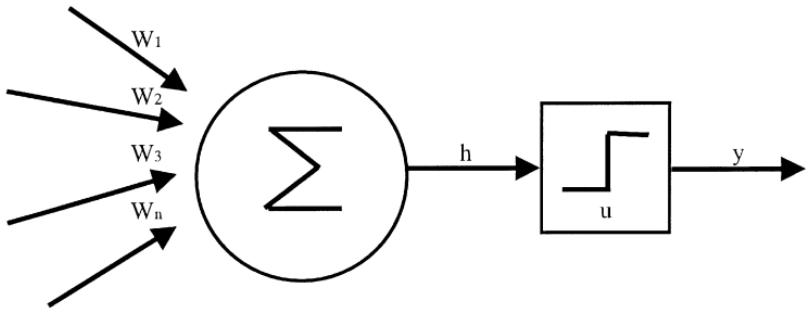

Figure 1 – An example of a neural network (Mair et al., 2000)

output becomes 1, otherwise, it is 0. If an output is incorrect, a process called backpropagation starts. In this process, the output is corrected by adjusting the weights. In this way, the networks learn from a dataset.

Sarkar, Ghalia, Wu, and Bose (2009) applied a neural network to predict fiber diameters by using different inputs. In this case they use a *multilayer network* as proposed by White (1992). A multilayer network has hidden layers between the original input and the final output variable(s). These hidden layers are functions that use the previous inputs to create an intermediate output node, which can be used as an input for another hidden layer or for the final output variable. Within one hidden layer there can be many layer nodes. It is also hard to tell how many layers and nodes a neural network should have because in the end the neural network determines what happens in the layer nodes within the layer. Sarkar et al. (2009) determine the number of nodes and layers by conducting experiments. The results were 12 nodes in the first hidden layer and 7 nodes in the second one. The final neural network can be seen in Figure 2.

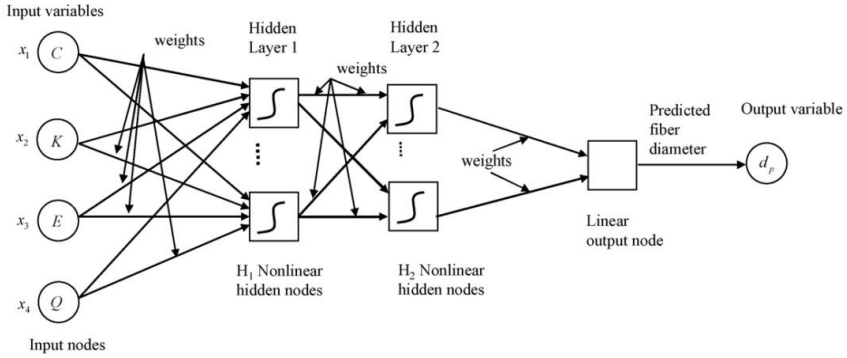

Figure 2 – An applied neural network with hidden layers (Sarkar et al., 2009)

Le Cun et al. (2012) have some recommendation for applying a neural network. For instance, shuffling the dataset helps the network to learn faster from unexpected samples (LeCun et al. 2012) and normalization of the presented input data can also increase the learning process. It is well-known that, the dataset should be split into a training set and a test set, in order to avoid overfitting, i.e., the neural network learns too much from the dataset in a sense that it also learns from outliers and noise instead of creating a general applicable prediction model. Furthermore, White (1992) and LeCun et al. (2012) state the use of too many parameters and too many layers can also lead to overfitting.

Another prediction model is introduced by Van Urk, Mes and Hans (2013). They use a prediction model for an application, which is somewhat similar to our problem, namely the development of a decision support application for the Dutch Aviation Police and Air Support unit for routing their helicopters in anticipation of unknown future incidents. Their research is similar as it involves a forecasting method and a routing method that maximizes the likelihood of being close to a future crime. Even though finding turtles and finding crime are not quite the same, the principle can be applied here as well. The second part of their research also deals with a kind of TOP but they combine it with a Location Covering Problem (LCP), as the helicopters have to cover certain areas to intervene quickly in case of emergencies. For us, the LCP is not relevant, as we do not deal with that kind of emergencies. More interesting, however, is the first part regarding the forecasting. In order to predict future crime intensity, they use a forecast based on the moment of the day, days of the week, and months of the year, which have an impact on the crime rates. They convert every past incident in order to use this information for future predictions. To this effect, they use two conversion factors, namely the FactorMonth (month, hour) and the FactorWeekday (weekday, hour). Additionally, they apply generalization techniques because they assume that an incident at one specific location and time is similar to the neighboring areas and some time periods around the incident. This model with some modifications can be used for this research as well. The incidents can be replaced by the number of saved turtles and it would be required to check whether the generalization also applies in our case. A

similar approach is discussed in the master thesis of van Hal (2015). In this research, the forecasting method is based on the fact that the relative distribution of incidents regarding the Netherlands does not depend on time. For that reason, the forecasting method is split into a time problem, which is solved by linear regression with different time-related factors, and a space problem, which is solved by means of the kernel density method.

2.4 Conclusion

In this chapter, we concluded that our proposed problem is too complex to be solved with an exact algorithm within reasonable time. Therefore, we presented possible heuristics that were applied to the original TOP without multiple visits:

- Tabu search embedded in an adaptive memory procedure
- Ant colonization optimization
- Simulated annealing
- Variable neighborhood search
- Path relinking heuristic with a greedy randomized adaptive search procedure
- Iterated local search heuristic

Furthermore, we discussed different options that could be used to predict the expected number of saved turtles, such as regression models, neural networks, and other forecasting models that were applied to similar problems.

3 Prediction method

For our prediction model, we recommended to apply a neural network model because neural network models are very efficient in solving different regression models. This chapter presents a possible neural network, which can be applied to a regression problem, such as the turtle saving problem. To design a neural network, different choices have to be made. The different design choices and conclusions that we made are discussed throughout this section.

3.1 The loss function

The loss function is essential to the network, as it learns by minimizing this function. We focused on two options, which are the absolute mean error and the mean squared error. Both are easy to interpret. A drawback of the mean squared error is that it weighs outliers a lot. Other loss functions have the disadvantage that they not as easy to interpret and do not allow 0 as an output or require a time series model.

3.2 The dataset

Le Cun et al. (2012) state that shuffling the dataset helps the network to learn faster from unexpected samples (LeCun et al., 2012). Furthermore, the dataset should be split, for example into a training set (80%) and a test set (20%), in order to avoid overfitting, i.e., the neural network learns too much from the dataset in a sense that it also learns from outliers and noise instead of creating a general applicable prediction model. Basheer and Hajmeer (2000) describe that it is also good to have an evaluation set next to the training and test set in order to see how well the generalization of the network works. This is something what we experienced because although the train set is new to the network, the test set was still very similar to the train set. A third split might help to see what happens if, for instance, the neural network is applied to a new and unknown cluster.

3.3 Choice and presentation of inputs and output

Usually the outputs are not adjusted and presented in their original form. Inputs should be different variables that have an impact on the output. They should either be a number within a certain scale or a cluster that represents a certain group (for instance: weekends, sunny days, etc.). In some cases normalization of the input variables can be useful to speed up the learning process of the neural network because the variables become easier to understand for the network. We propose 3 different kinds of normalizations:

- Value – average of all values. This normalization returns the values in such a way that the average is 0.
- Value/average of all values. This normalization only works for positive values. Returns small positives values, where the average is 1.
- Minmax normalization. Min is the minimal value of the dataset. Max is the maximal value of the dataset. The value of the variables is transformed by means of the following formula:

$$\text{Normalized value } = \frac{Value - Min}{Max - Min}$$

3.4 Number of hidden layers and nodes within a layer

The design of the network regarding the number of hidden layers and the nodes within a layer provides infinitive possibilities. White (1992) and LeCun et al. (2012) state the use of too many parameters and

too many layers can lead to overfitting. For our regression problem, we considered therefore 1 or 2 hidden layers. The number of nodes in the hidden layers is based on experiments.

3.5 The activation function of a layer

The activation function of a layer determines the output of the hidden layers. The performance of the activation function is based on the different input variables and the final desired output. For our regression problem, the “Rectified linear units” shows the best performance. The definition of this function is: F(x) = max(x, 0).

3.6 Computing inputs

While computing inputs for our routing algorithm with the proposed neural network, we notice some strange values. These values are, for instance, negative values, which should not be possible, or extremely high values. It seems that these values occur whenever there is a gap in the regarding the location or time. Despite these gaps we hoped that the generalization of the neural network would compute reliable numbers anyways. Basheer and Hajmeer (2000) describe that it is good to have an evaluation set next to the training and test set in order to see how well the generalization of the network works. In our case, this could be done by taking some specific observations (e.g., a few beaches or one specific hour) out of the original data set. These excluded observations would form the evaluation set. Afterwards, the remaining data set can be split into the test and training set. By doing so, the prediction model, which is trained on the training set, can be tested for overfitting on the test set and its generalization capabilities can be evaluated by means of the evaluation set.

3.7 Conclusion

In this chapter, we have shown how a neural network can be applied to such a regression and which variables may be interesting to consider. For training our neural network, we used a test and training set to optimize the design of our training network based on the results of the test set. We have learned that adding an evaluation set to the test and training set, can be a valuable supplement to test the generalization capacity of the network.

4 Routing algorithm

In this chapter, we design the routing algorithm that we developed for the proposed turtle rescue problem. Section 4.1 explains the notation that is used throughout this chapter, Section 5.2 deals with the objective function of the algorithm in order to evaluate different solutions, and Section 5.3 describes how the algorithm works.

4.1 Notation

This section explains the notation that we use throughout this chapter to describe the routing algorithm. Note that we consider the entire time span of one day to be from 9.00, which is equivalent to 9 am of the scheduled day, until 28.00, which is equivalent to 4 am of the following day. The routing algorithm provides a planning that schedules visits of the beaches, i.e., that a beach is inspected at a certain time.

Every visit is defined by the number of the beach j (1…J), the region of the beach (1…A), the day of the planning period n (1…N), the weekday d (1…7), the starting time t (between 9.00 and 28.00), the finish time t_f (between 9.00 and 28.00), the week number w (1…W), the year y (1…Y), and the number of the competitor m (1…M). Therefore, every visit is denoted as $v_{j,a,n,d,t,t_f,w,y,m}$. For the purpose of simplicity, we denote the visits in this chapter as $v_{j,n,t,m}$ because it is possible to derive the region from beach j and the weekday d, the week number w, and the year y from the planned day n, and the finish time t_f from the start time t. We denote this visit also as current visit because later in this chapter we also need information about the *prior visit* and the *sequential visit*. The prior visit describes the last time that beach j was visited before the current visit $v_{j,n,t,m}$. The beach j has to be the same in this case but the competitor m and planned day n not necessarily. Likewise, the sequential visit describes the next visit of beach j after the current visit. We use an additional variable x in order to denote whether we consider the current (x=0), prior (x=-1), or sequential visit (x=1). Consequently, we denote every visit as $v_{j,n,t,m,x}$. We provide an overview of this notation in Table 2.

Index	**Definition**	**Range**
j	number of the beach	1…J
a	region	1…A
n	day of the planning period	1…N
d	weekday	1…7
t	starting time	between 9.00 and 28.00
t_f	finish time	between 9.00 and 28.00
w	week number	1…W
y	year	1…Y
m	number of the competitor	1…M
x	visit information	-1,0,1

Table 2 – Notation table of the indices of a visit

Whenever we require the value of a certain index of a visit, we do this by putting $v_{j,n,t,m,x}$ in its index. For instance, the finish time of the visit of beach 15 by competitor 3 on day 4 at 12.00 can be expressed as $t_f\ v_{15,4,12.00,3,0}$. The finish time of the visit that is prior to that one can be described as

$t_{f\ v_{15,4,12.00,m,-1}}$. Let us say that the prior visit started at 9.00 and was done by competitor 1, then we can denote the finish time also as $t_{f\ v_{15,4,9.00,1,0}}$ (the prior visit can also be denoted as a current visit).

4.2 Priority score

In this section, we introduce our objective function, the priority score that determines the added value of visiting a beach. As stated in the beginning, the number of saved turtles should be maximized but also the following two objectives should be taken into account.

1. Save a certain number of turtles in every region in every month.
2. Visit every beach every three days.

For this purpose, we introduce the two factors: the goal factor $T_{v_{j,n,t,m,0}}$ and the day visit factor $V_{v_{j,n,t,m,0}}$. The goal factor focusses on the first objective. It is computed by a function that returns a value between 0 and 2. If the value is lower than 1, the pre-set goal is already reached. If the goal is not reached yet, the factor becomes a value higher than 1, such that the node attracts more visits. The factor is computed as follows:

$$G_{v_{j,n,t,m,0}} = 2 - \min(2, \frac{\text{number of saved turtles}}{\text{targeted number of saved turtles}})$$

The visit day factor focuses on the second priority and is also computed by means of a function. Since this objective is less important, it always returns values equal or above 1. Therefore, it cannot make a node less attractive. The factor is 1 if the node is visited the same day. When the number of days that the node has not been visited grows, the factor starts to increase slowly but at some point it will overrule all other aspects, such that the node is visited.

Finally, we compute the priority score by multiplying the number of saved turtles, which is denoted as $P'_{v_{j,n,t,m,0}}$ (see Section 5.4.2), by the goal factor and visit day factor. Therefore, the objective function for all days and competitors is expressed as:

$$max \sum_{n=1}^{N} \sum_{m=1}^{M} P'_{v_{j,n,t,m,0}} * G_{v_{j,n,t,m,0}} * V_{v_{j,n,t,m,0}} .$$

This priority score will be essential for our routing algorithm.

4.3 Routing algorithm

We chose to construct the route planning by means of an ant colonization optimization algorithm (ACO) because the benchmark of Ke et al. (2007) shows that for the team orienteering problem the ACO algorithm shows similar or better results in terms of the objective function in comparison with other algorithms but with a better computational time. In addition, we wanted to apply a constructive metaheuristic because local search metaheuristics are more difficult to apply to this generalization of the TOP due to the multiple visits and time-dependency. However, we do apply a 2-Opt swap, which exchanges the two previously scheduled visits of the current competitor, if the saved travel time is above a certain Swap Treshold and the solution is not affected negatively. We define the solution to be not affected negatively if the travel time reduction in percentages is bigger than the decrease of saved turtles and if the time after the swap must be smaller than the time before the swap. The rationale behind this is that due to the saved time, it is possible to earn the loss of saved turtles back. This beach swap is similar to the work of Verbeeck et al. (2014a). Unlike Verbeeck et al. (2014a), however, we apply this 2-Opt swap while constructing a solution whenever a beach is scheduled. Another swap that we

apply during the construction of a solution is a stopping point swap. Because it is not known which beach the competitor visits after going to a stopping point, it can happen that not the best stopping point is chosen in terms of travel distance. Figure 3 shows an overview of the general concept that explains how our algorithm works. We denote a day planning as one planning with all routes of all competitors for one planned day, where M is the total number of competitors. The algorithm creates such a day planning for every day, starting from day n until a certain day N. For every day, our algorithm performs many iterations, where the number of iterations I is a pre-determined parameter. One iteration contains one day planning. Every planned day starts by first setting the pheromone values of all arcs to a pre-determined initial value. Afterwards, I iterations are performed. At the end of each iteration, the pheromones of the arcs will be updated to create a learning effect. Since this update is a core process of our algorithm, it is highlighted in Figure 3 and will be further discussed in Section 4.3.3. For every day, we keep the iteration that led to the best result. We send our ants, which are equal to the competitors, sequentially as proposed in the algorithm of Montemanni and Gambardella (2009) and Ke et al. (2007). Ke et al. (2007) also tried different methods but the sequential approach seemed to perform the best.

For every day in range (n,N):
- → Replace pheromone values by pre-determined initial pheromone values
- For every iteration (1, I):
  - For every ant (1,M):
    - → Create a route planning
  - → **Update Pheromones**

Figure 3 – General concept of our ACO routing algorithm

Before going into detail on how the route of every ant is build, we denote the following terms:

- StoppingTime: is the starting time of the next stop
- StoppingFlexibility: is the tolerance of being early or late, which is at the moment 15 minutes
- DayStartTime: the time at which the day starts and the ant leaves the depot
- DayFinishTime: the time at which the day finishes and the ant has to return to the depot
- Beach (j): the next assigned beach
- TravelTimeToBeach: the travel time from the current beach to the new beach
- TravelTimeToStoppingPoint: the travel time to go back to the nearest stopping point
- TravelTimeToDepot: the travel time from the current beach to the depot
- CurrentTime: the time of the decision
- ServiceTime: the time that is needed to inspect a beach
- ShiftChange: the time when the first competitors shift is over and the sequential competitor arrives to replace the first one

The routing algorithm takes into account all time restrictions, such as stopping times, start and finish time of the day, and the shift change. Therefore, the time needs to be updated after every step. It is also required to update the number of saved turtles per region to determine the goal factor. Figure 4 shows the route planning of an ant. This route planning is performed differently in the first iteration. We choose to build a greedy solution in the first iteration and use the ACO algorithm afterwards. In Figure 4, there are two more core decisions of our algorithm that are highlighted in Figure 4 that work differently in the first iteration. We discuss these further in Section 4.3.2 and Section 4.3.3.

For every ant:
Start
Depending on the time, the ant chooses one of the following options:
If (ShiftTime ≤ CurrentTime < ShiftTime+StoppingFlexibility):
→ Ant stops for 15 minutes
→ Update Time (go back to start)
Else if (StoppingTime-StoppingFlexibility ≤ CurrentTime+TimeToStoppingPoint < StoppingTime+StoppingFlexibility):
→ Ant goes to the nearest stopping point
→ Update Time (go back to start)
Else if (FinishTime-StoppingFlexibility ≤ CurrentTime+TimeToDepot)
If ant is at stopping point (including depot):
→ Ant is **distributed to a region** and **chooses a beach** within that KPI area and adds it to the route
→ Consider stopping point swap
→ Update time and KPI matrix (go back to start)
Else:
→ Ant **chooses a beach** out of a set of nearest beaches (set of beaches is explanation in Section 5.4.2.1)
If (previous two visits are no stopping points and no shift changes) and (decrease of travel times is bigger than no decrease of the objective function):
→ Switch previous two beaches
→ Update time and KPI matrix (go back to start)
Else:
→ Update Time and KPI matrix (go back to start)
Else:
→ Ant goes back to depot and finishes the route
→ Set time to DayStartTime (go to next ant)

Figure 4 – Route planning of one ant

4.3.1 Distribution strategy

Our distribution strategy allocates a certain number of competitors to every region whenever an ant leaves the depot or the stopping point. Based on this outcome, the ant is send to a region. The distribution is based on two things: the goal factor and the *area size factor*. The area size ratio is the ratio how large the goal of the region is in comparison to the sum of all goals. We multiple the area size ratio with the goal factor in order to compute an adjusted goal factor. By dividing this adjusted goal factor by the sum of all adjusted goal factors, the relative importance of a region is computed. Finally, this relative importance is multiplied by the total number of competitors to compute how many competitors should be send to a region.

4.3.2 Choosing a beach

The next chosen beach depends on two things: the criteria of choosing a beach and the set of beaches from which the next beach is chosen. Both depend on the current iteration. In Section 5.4.2.1, we

explain how the sets of considered beaches is determined. Section 5.4.2.2 introduces our restoring function, which is essential in dealing with multiple visits, Section 5.4.2.3 describes our desirability function and Section 5.4.2.4 the probability function for our ACO algorithm.

4.3.2.1 The set of beaches

Whenever an ant chooses the next beach out of a certain set of beaches, there are three possible sets that can be considered:

1. All beaches within a region.
2. All beaches that can be reached within a pre-set travel distance, which is denoted as travel distance restriction.
3. All beaches.

The ant chooses one of the beaches of Set 1, whenever an ant leaves a stopping point. Set 2 is considered, whenever the ant is not at a stopping point. The reason for choosing beaches within a pre-set travel distance is that the competitors have a limited average speed and cannot travel from every beach to any other one. The pre-set travel distance should therefore depend on the average speed of the competitors. Whenever no feasible beaches can be found in Set 1 or Set 2, the ant considers Set 3 as a kind of backup set.

In the first iteration, the ant chooses the beach based on the result of the greedy function (see Section 5.4.2.2). Thereafter, it considers both the greedy function and the pheromone trails, as explained in Section 5.4.2.3. There is only an exception for Set 3. Whenever, the backup set is considered, the choice is only based on the greedy function, because we do not update the pheromone trails of all possible arcs to safe computation time.

4.3.2.2 Restoring function

This restoring function is essential for our algorithm to deal with the problem that we introduced. The fact that one node can be visited multiple times a day leads to the question when it makes sense to return to a visited node. For that purpose, we introduce the restoring function R(t), which returns a value between 0 and 1. This value is a percentage that determines to what extent the expected reward at a node is restored. The restoring function starts after a node has visited and increases exponentially with time until it reaches 1. This means the node is restored in a sense that the two visits are not inter-related anymore. For example it is predicted that at 10 am 3 turtles can be found and at 12 am 4 turtles can be found. In that case 3 turtles could have possible stayed for these 2 hours. The expected number of turtles at 12 am is then 1+3*R(2). If the node is already restored (R(2) = 1), then the expected number of turtles is 4. If not, the expected number is between 1 and 4.

4.3.2.3 The desirability function

Our desirability function is a greedy function that divides the priority score of a visit by the time that is needed to visit the beach. It is an adjusted version of the one that Ke et al. (2008) are using.

Concerning the travel time from a stopping point b to the next beach j, we use the accounted travel time, which is raw travel time reduced by a pre-set travel time reduction parameter. The rationale behind it is to distribute the beaches farther away from the stopping points. However, the accounted travel time may never exceed another pre-set parameter, namely travel time maximum. The purpose of this maximum is to restrict the travel time, whenever the ant chooses out of all possible beaches (Set 3).

However, the time does not only include the accounted travel time from beach i to beach j but also the service time from beach j and the travel time from beach j to the next stopping point b (or the depot

depending on the shift). The latter, however, is not always important. When a shift starts, the competitor should or even must distance itself from the stopping point in order to avoid that it only drives close to the stopping point. The closer the time gets to the stopping time, the more important the travel time to the stopping point gets. Therefore, we use a progress factor $p_{v_{j,n,t,m,0}}$, which determines how strong the travel time weighs within the desirability function. Whenever the ant leaves the depot or stopping point, the value is close to 0. Then, it increases up to 1, when it should return to the depot or stopping point. The desirability function of a visit is denoted as $\eta_{v_{j,n,t,m,0}}$ and can be denoted as:

$$\eta_{v_{j,n,t,0}} = \frac{\Delta_{P'_{v_{j,n,t,m,0}}} * T_{v_{j,n,t,m,0}} * V_{v_{j,n,t,m,0}}}{c_{i,j,t,d} + s_{j,t,d} + p_{v_{j,n,t,m,0}} * c_{j,b_j,t,d}}.$$

where $\Delta_{P'_{v_{j,n,t,m,0}}}$ is the increase in terms of rewards, which already takes the restoring function into account, and $c_{i,j,t,d}$ is the accounted travel time from beach i to beach j, at time t, on weekday d, $s_{j,t,d}$ is the service time at beach j, at time t, on weekday d, and $c_{j,b_j,t,d}$ is the travel time from beach j to the to the closest stopping point from beach j, which is b_j.

As mentioned in Section 5.4.2.1, in the first iteration only the desirability function is used to choose the next beach from Set 1 or Set 2. At the end of the first iteration, the pheromone trails are updated the first time as we describe in Section 5.4.3. Afterwards, the probability function of the ACO algorithm is applied to find better solutions than the first greedy solution as we explain in Section 5.4.2.3.

4.3.2.4 The probability function of the ACO algorithm

In the ACO approach the ant does not always choose the "most desirable" solution but goes probabilistically from beach i to beach j, whereby node j is from a set of considered beaches L(i), as determined in Section 5.4.2.1. The choice depends on the probability of node j, which is based on two factors: the desirability $\eta_{v_{j,n,t,0}}$ and the pheromone trails $\tau_{i,j}$. The pheromone trails depend on the pheromone value on the arc between beach i and j, which is updated after every iteration as explained in Section 5.4.3. The probability p_{ij} is for all beaches determined as follows:

$$p_{ij\,j \in \mathrm{N(i)}} = \frac{\tau_{i,j}^{\alpha} * \eta_{v_{j,n,t,0}}^{\beta}}{\sum_{l \in \mathrm{N(i)}} \left(\tau_{i,l}^{\alpha} * \eta_{v_{l,n,t,0}}^{\beta}\right)},$$

where α and β are used to control the importance of the pheromone trails and the desirability.

All beaches j are ranked according to their probabilities. Afterwards, a random number is generated with a uniform distribution between 0 and Q_0. Q_0 is a randomness parameter, which determines how big the generated numbers can be. If the random generated number is smaller than the p_{ij} of the first beach in the ranked set, beach j is chosen, otherwise, a new number is generated and compared to p_{ij} of the second beach and so on. Q_0 decreases after every iteration by the decrease parameter D_{Q_0}, such that the beaches with the most probability are more likely to be added to the route. The rationale behind it is that in the beginning many different solutions are explored but towards the end we want the ants to choose the best option in terms of the computed probability p_{ij}.

4.3.3 Update of pheromone trails

The first iteration of every day creates a solution of the route planning based on the greedy algorithm. The objective function of this day planning is saved as $Solution_{Greedy}$. The greedy solution serves as a benchmark solution, which we strive to exceed. The performance of the day planning of all iterations afterwards are measured by comparing it to the greedy solution: $Performance_{Iteration}$ =

$\frac{Solution_{Iteration}}{Solution_{Greedy}}$. The solution with the best performance is saved as $Solution_{Best}$ and $Performance_{Best}$. Since the greedy solution is the first created solution, it is the first best solution. We strive to maximize the best performance by updating the visited arcs with a pheromone trail after every iteration. The update of the pheromone trail is essential to the algorithm as it teaches the ants, which combination of beaches worked well in the past iterations. This section introduces three different update strategies.

4.3.3.1 Update Strategy 1

Due to our generalization of the TOP that allows to visit beaches multiple times and also includes other additional constraints, there is no ACO algorithm in the literature that tackles the same problem. However, we decided to derive our first updating strategy from the work of Montemanni and Gambardella (2009), who develop an ACO algorithm for the TOP with time windows. In their algorithm, only the ant that produced the best solution since the beginning of the computation, which we denoted as $Solution_{Best}$, is allowed to leave a pheromone trail. The reason behind it is that the best route is memorized, and in the future, ants will generate new (and hopefully better) solutions that are similar to this route. For that reason, we also update the pheromone of the visited arcs, only whenever a best solution is achieved. If an arc is visited more than once, it only counts as one visit. The pheromone trails are denoted as $\tau_{i,j}$, where i and j present the arc between beach i and j. The initial value is denoted as τ_0. The updating rule is the following:

$$\tau_{i,j} = (1 - \rho) * \tau_{i,j} + \rho * Performance_{Best},$$

where p regulates the strengths of the pheromone that is left by the best solution. After the first iteration, this update is applied the first time. In this case, the initial value $\tau_{0_{i,j}}$ determines the attractiveness of the arcs that have not been visited by the greedy solution in comparison with the ones that have been. Since the $Performance_{Best}$ of the greedy solution is per definition equal to 1, we choose τ_0 to be smaller than 1 in order to attract more ants to the greedy solution. The exact value will be determined in Chapter 6.

Moreover, during the construction of a route, every ant decreases the pheromone trails of the arcs that is has used to prevent that arcs are visited too many times and to stimulate the exploration of new solutions. The rule is determined by:

$$\tau_{i,j} = (1 - \psi) * \tau_{i,j} + \psi * \tau_0,$$

where ψ is the evaporation parameter that regulates the decrease of the pheromone trace. We do not want the ants to visit beaches many times because it decreases the exploration, plus it decreases also the performance of the solution as discussed in Section 5.4.2. If there is no improvement after the iteration, the pheromones are not updated but restored, meaning that the decrease during the solution building does not apply for the next solution.

4.3.3.2 Update Strategy 2

For this strategy, we slightly adjusted Strategy 1. In this strategy, the arcs are updated after every solution instead of updating the arcs only after an improvement of the $Solution_{Best}$. After every iteration all visited arcs are updated according to the formula:

$$\tau_{i,j} = (1 - \rho) * \tau_{i,j} + \rho * Performance_{Iteration}.$$

4.3.3.3 Update Strategy 3

Strategy 3 is another variation of Strategy 1. The difference is that the ants are considered separately. Not the best solution of all ants together counts but the best solution of every ant separately. The rationale behind this is that the allocation to a region should be the same in most cases for all iterations. When the same ant is send to the same region every iteration, we can measure the objective function of this ant. Every ant has therefore a separate performance that is compared to the greedy solution of that ant. By doing so, we also increase the pheromone trails of arcs that have been visited by a single well-performing ant even though most ants performed worse. If an arc is visited by more than one ant that achieved a personal best score, then the one with the higher score may set the trail. Regarding the update of the best solutions or all solutions, we can either use the updating rule as described in Strategy 1 or 2 for this strategy.

5 Results of the routing algorithm

In this chapter, we focus on the results of our ACO algorithm. We considered a case with 320 beaches within 10 regions. We assumed that it would be reasonable to create routes for 12 competitors (also denoted as ants). When making routes for 12, we had some difficulties to find better solutions than the solution of the greedy algorithm (denoted as greedy solution), which is the solution of the first iteration. After trying different parameters and update strategies, we wonder whether the number of competitors has an impact on the performance of the ACO algorithm. For that reason, we considered making a route for only one competitor, and indeed we received good results with the following parameter values (parameters are introduced in Section 4.3.2.4 and Section 4.3.3):

- Update Strategy 1
- $\rho = 0.90$
- $\psi = 0.95$
- $Q_0 = 0.4$
- $D_{Q_0} = 0.1$
- $\tau_0 = 0.8$
- $\alpha = 1$
- $\beta = 2$
- Number of iterations = 10

The outcome for one competitor is shown in Table 3.

Day	Iteration	Improvement compared to the priority score of the greedy solution
1	1	0%
1	2	7%
1	3	1%
1	4	15%
1	5	9%
1	6	4%
1	7	5%
1	8	18%
1	9	11%
1	**10**	**20**%

Table 3 – Results of the ACO algorithm for one competitor

We derive from this table that the priority score increases with the number of iterations. After 10 iterations, the solution is already improved by 20%. Therefore, we conclude that the ant is indeed learning and that our ACO algorithm works at least for one competitor.

Unfortunately, when applying the same parameters to more competitors, we see less and less improvement. For 7 competitors, it starts that we see less and less improvements with the presented parameter settings. Figure 5 shows the improvement in comparison with the greedy algorithm for 1, 2, 3, and 7 competitors in 10 iterations.

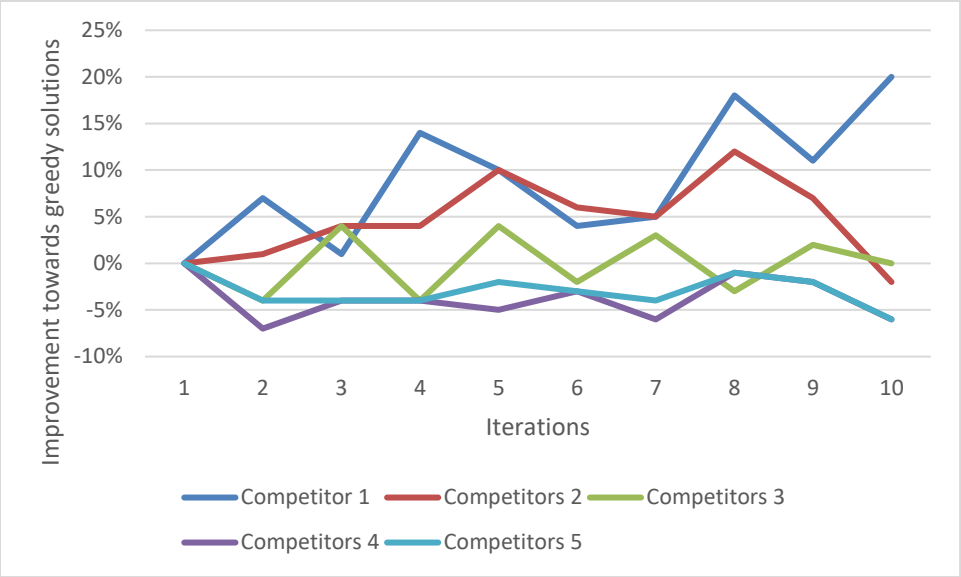

Figure 5 the ACO algorithm for 1, 2, 3, 7, and 12 competitors

For 12 competitors, it is sometimes possible to find an improvement; however, it takes a lot of time and only improves the solution by roughly 1%. For many competitors the results retrieved by our third strategy, which updates the pheromones separately for all competitors, seems to perform a bit better. However, still not good enough to significantly improve the solution. Therefore, we investigate what happens when (too) many competitors are used.

One logical explain could be that with more deployed competitors, also more competitors are deployed to the same region. When many ants share the same region, the learning effect that always applies for one single ant gets more complicated. For instance, in the past the first ant has learned a new route that improves the results. However, it might happen that the second ant changes its route within this iteration and visits some nodes that the first ant would have travelled to. Therefore, the learning effect of the first ant becomes in this example useless.

Furthermore, there could be another reason that leads to a decrease of the learning effect. If a beach must not be visited more than once a day and 12 competitors are scheduled, every competitor can visit 26.67 (320/12) beaches on average. With our parameter settings and inputs, every competitor visits around 45-50 beaches a day. This means that ants are forced to visit beaches multiple times a day because the map gets too saturated in terms of visited beaches. For that reason, our algorithm does not work for 12 competitors if a beach must not be visited multiple times a day. If we assume that every competitor visits on average 45 beaches a day, then only 7.11 competitors (320/45) will be required to manage to visit all beaches. After 7 competitors, one could say that the map gets "oversaturated" and therefore beaches have to be visited more than once a day (note that ants choose also to visit beaches many times a day because they are just desirable and not because there is no other choice). This in combination with the fact that the desirability function takes the oversaturation more into account than the pheromones, could be an explanation why the ACO algorithm shows worse results for more than 7 competitors. The stability function has an impact on how many times an attractive beaches is visited a day. Therefore, we want to investigate whether a different stability function that leads to fewer multiple visits of the beaches increases the performance of the ACO algorithm for 12 ants. For this purpose, we choose the following linear stability function: $S(\Delta_t) = \max(1 - 0.1\Delta_t, 0)$. This function leads to less beaches visited multiple times a day. During our experiments, we managed to improve the greedy solution by 3% after 10 iterations with the following parameter setting:

- Update Strategy 3
- $\rho = 0.95$

- ψ = 0.95
- Q_0 = 0.4
- D_{Q_0} = 0.1
- τ_0= 0.8
- α = 1
- β = 2.5
- Number of iterations = 10

As we see that the ACO algorithm can improve the greedy solution when using another stability function, we conclude that the stability function has an impact on the performance of the ACO algorithm. Since there is a reason for our stability function, changing it is not an option. We assume that this improvement that occurs due to changing the stability function has the same cause that we experienced before with less competitors, namely that there are less multiple visits of beaches a day. Apparently the ACO algorithm performs better when the problem is more similar to the original TOP without multiple visits than our proposed generalization with multiple visits. It seems that the pheromones of the ACO algorithm cannot deal with the multiple visits as good as our greedy algorithm. Therefore, we conclude that the ACO algorithm might not be the best choice to deal with this new generalization. Since the entire problem especially with the multiple visits is very time-related, it might help to apply time-dependent pheromones, which was already done by Jiang, Chen, Ma, and Deng (2011). However, even though the time-dependency would be included in the learning effect, it is not guaranteed that time-dependent pheromones would really lead to an improvement regarding the multiple visits. Considering the greedy algorithm, the algorithm could be improved by adding a saturation factor that determines how many beaches have been visited already in that area around the considered beach at a certain time.

5.1 Conclusion

We concluded in this chapter that even if the ACO algorithm leads to a better result than the greedy algorithm, the improvement is very little for the usual number of deployed competitors (smaller than 1%). If we want to apply the ACO algorithm with 10 iterations, it means that the computation time for one route is more than 10 times larger (10 iterations + 10 times a pheromone update). Therefore, we chose in our research to only apply the greedy algorithm.

6 Conclusion

In this summary of our research, we discussed the “turtle rescue problem”, in which there is an organization that wants to maximize the number of saved sea turtles. Every day, a fleet of homogeneous competitors leaves from one depot to go to different beaches. At every beach, they can find a certain number of turtles. A beach can be visited multiple times on one day. Next to some additional targets, it is the main objective to maximize the number of saved turtles. We defined this problem as a new generalization of the TOP, namely the TD-PTOPMVMC. It was the objective of our research to find a method that predicts the number of rescued turtles at a beach and to develop a routing algorithm that solves this problem.

Regarding the objective of our research, we discussed different possibilities in our literature study. Finally, we discussed the implementation of a neural network, which can be applied to such a regression problem. For training our neural network, we used a test and training set to optimize the design of our training network based on the results of the test set. We have learned that adding an evaluation set to the test and training set, can be a valuable supplement to test the generalization capacity of the network. This is also discussed by Basheer and Hajmeer (2000). In our case, this could be done by taking some specific observations (e.g., a few beaches or one specific hour) out of the original data set. These excluded observations would form the evaluation set. Afterwards, the remaining data set can be split into the test and training set and the results of trained model can be tested for overfitting on the test set and the generalization capabilities of the neural network can be evaluated based on the evaluation set.

Furthermore, we chose to apply an ACO algorithm. Unfortunately, we learned during our experiments that the ACO algorithm seems to have some troubles when many competitors are deployed. One possible reason is that with too many competitors, the learning effect of the ants is decreased due to the fact that another ant “steals” the visits on the desired path. Another reason is that the ACO algorithm has difficulties when nodes are visited multiple times. The more this happens, the less the problem is similar to the original TOP. We concluded that our proposed ACO algorithm is not the best choice to solve our introduced generalization of the TOP. Since the entire problem especially with the multiple visits is very time-related, it might help to apply time-dependent pheromones, which was already done by Jiang, Chen, Ma, and Deng (2011). However, even though the time-dependency would be included in the learning effect, it is not guaranteed that time-dependent pheromones would really lead to an improvement regarding the multiple visits. Apart from that, in the future other constructive heuristics (e.g., adaptive search) could be applied to this problem.

7 References

Archetti, C., Hertz, A., & Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, *13*(1), 49-76.

Awerbuch, B., Azar, Y., Blum, A., & Vempala, S. (1998). New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, *28*(1), 254-262.

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, *43*(1), 3-31.

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, *34*(3), 209-219.

Boussier, S., Feillet, D., & Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4OR: A Quarterly Journal of Operations Research*, *5*(3), 211-230.

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The competitor routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*, 300-313.

Butt, S. E., & Ryan, D. M. (1999). An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, *26*(4), 427-441.

Chao, I., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. European Journal of Operational Research, 88(3), 464-474.

Cordeau, J. F., Gendreau, M., Hertz, A., Laporte, G., & Sormany, J. S. (2005). New heuristics for the competitor routing problem. In *Logistics systems: design and optimization* (pp. 279-297). Springer US.

Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation science*, *39*(2), 188-205.

Gavalas, D., Konstantopoulos, C., Mastakas, K., & Pantziou, G. (2014). A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, *20*(3), 291-328.

Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., & Vathis, N. (2015). Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research*, *62*, 36-50.

Gendreau, M., Laporte, G., & Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, *106*(2-3), 539-545.

Golden, B. L., Laporte, G., & Taillard, É. D. (1997). An adaptive memory heuristic for a class of competitor routing problems with minmax objective. *Computers & Operations Research*, *24*(5), 445-452.

Graham, S. M., Joshi, A., & Pizlo, Z. (2000). The traveling salesman problem: A hierarchical model. *Memory & cognition*, *28*(7), 1191-1204.

Hochbaum, D. S. (1995). A nonlinear knapsack problem. *Operations Research Letters*, *17*(3), 103-110.

Jiang, B. B., Chen, H. M., Ma, L. N., & Deng, L. (2011). Time-dependent pheromones and electric-field model: a new ACO algorithm for dynamic traffic routing. *International Journal of Modelling, Identification and Control*, *12*(1-2), 29-35.

Ke, L., Archetti, C., & Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, *54*(3), 648-665.

Kumar, M., Husian, M., Upreti, N., & Gupta, D. (2010). Genetic algorithm: Review and application. *International Journal of Information Technology and Knowledge Management*, *2*(2), 451-454.

Larsen, R. J., & Marx, M. L. (2012). *An introduction to mathematical statistics and its applications* (Vol. 5). Englewood Cliffs, NJ: Prentice-Hall.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9-48). Springer Berlin Heidelberg.

Lin, S. W., & Vincent, F. Y. (2015). A simulated annealing heuristic for the multiconstraint team orienteering problem with multiple time windows. *Applied Soft Computing*, *37*, 632-642.

Mair, C., Kadoda, G., Lefley, M., Phalp, K., Schofield, C., Shepperd, M., & Webster, S. (2000). An investigation of machine learning based prediction systems. *Journal of Systems and Software*, *53*(1), 23-29.

Montemanni, R., & Gambardella, L. M. (2009). An ant colony system for team orienteering problems with time windows. *Foundation Of Computing And Decision Sciences*, *34*(4), 287.

Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for competitor routing. *Journal of heuristics*, *1*(1), 147-167.

Sarkar, K., Ghalia, M. B., Wu, Z., & Bose, S. C. (2009). A neural network model for the numerical prediction of the diameter of electro-spun polyethylene oxide nanofibers. *Journal of materials processing technology*, *209*(7), 3156-3165.

Souffriau, W., Vansteenwegen, P., Berghe, G. V., & Van Oudheusden, D. (2010). A path relinking approach for the team orienteering problem. *Computers & operations research*, *37*(11), 1853-1859.

Tang, H., & Miller-Hooks, E. (2005). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, *32*(6), 1379-1407.

Tang, H., Miller-Hooks, E., & Tomastik, R. (2007). Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, *43*(5), 591-609.

Tricoire, F., Romauch, M., Doerner, K. F., & Hartl, R. F. (2010). Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, *37*(2), 351-367.

Van Hal, K. (February 2015). When and Where to Fly and Stand by (Unpublished master thesis). Department of Industrial Engineering and Management, University of Twente, Enschede.

Vansteenwegen, P. (2009a). Planning in tourism and public transportation. *4OR: A Quarterly Journal of Operations Research*, *7*(3), 293-296.

Vansteenwegen, P., Souffriau, W., Berghe, G. V., & Van Oudheusden, D. (2009b). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, *36*(12), 3281-3290.

Vansteenwegen, P., Souffriau, W., Berghe, G. V., & Van Oudheusden, D. (2009c). Metaheuristics for tourist trip planning. In *Metaheuristics in the service industry* (pp. 15-31). Springer Berlin Heidelberg.

Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, *209*(1), 1-10.

Van Urk, R., Mes, M. R., & Hans, E. W. (2013). Anticipatory routing of police helicopters. *Expert systems with applications*, *40*(17), 6938-6947.

Verbeeck, C., Sörensen, K., Aghezzaf, E. H., & Vansteenwegen, P. (2014a). A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research*, *236*(2), 419-432.

Verbeeck, C., Aghezzaf, E. H., & Vansteenwegen, P. (2014b, November). Solving the Stochastic Time-Dependent Orienteering Problem. In *MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation*.

White, H. (1992). *Artificial neural networks: approximation and learning theory*. Blackwell Publishers, Inc.