DEPARTMENT OF ENGINEERING MANAGEMENT

What makes a solution good?
The generation of problem-specific knowledge for heuristics

Florian Arnold & Kenneth Sörensen

UNIVERSITY OF ANTWERP
Faculty of Applied Economics

City Campus
Prinsstraat 13, B.226
B-2000 Antwerp
Tel. +32 (0)3 265 40 32
Fax +32 (0)3 265 47 99
www.uantwerpen.be

FACULTY OF APPLIED ECONOMICS

DEPARTMENT OF ENGINEERING MANAGEMENT

What makes a solution good?
The generation of problem-specific knowledge for heuristics

Florian Arnold & Kenneth Sörensen

RESEARCH PAPER 2017-003
JANUARY 2017

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
fax: (32) 3 265 47 99
e-mail: joeri.nys@uantwerpen.be

The research papers from the Faculty of Applied Economics are also available at www.repec.org (Research Papers in Economics - RePEc)

D/2017/1169/003

What makes a solution good? The generation of problem-specific knowledge for heuristics

Florian Arnold [1a], Kenneth Sörensen[a]

[a]*University of Antwerp, Departement of Engineering Management, ANT/OR - Operations Research Group*

Abstract

Heuristics are the weapon of choice when it comes to solving complex combinatorial optimization problems. Even though some research focuses on tuning a heuristic with respect to a certain problem, little research has been done to investigate structural characteristics of the problem itself. In this paper we argue that knowledge about a problem is highly valuable when it comes to designing efficient heuristics, and we show how it can be generated. With knowledge we hereby mean that we can define desirable structural characteristics of good solutions.

Our knowledge generation approach is based on data mining and we demonstrate its concept with the help of the most prominent combinatorial problem in Operations Research, the Vehicle Routing Problem. We define metrics to measure a solution and an instance, and generate and classify 192.000 solutions for various instances. With these metrics we are able to distinguish between optimal and non-optimal solutions with an accuracy of up to 93%. Furthermore, we reveal the most distinguishing characteristics of good VRP solutions, and use them to improve an existing heuristic.

Keywords: Data mining, Heuristics, Vehicle Routing Problem, Problem-specific knowledge

1. Introduction

Complex problems are entangled with our everyday life, usually without us noticing. We use transportation systems and rely on train and airport providers

[1]corresponding author. Email: florian.arnold@uantwerpen.be

to reliably schedule several thousand journeys a day. We rely on logistic providers to deliver our Amazon order among several thousand others in time, especially before Christmas. These kind of problems are omnipresent in a world with ever-growing complexity.

To master this growing complexity, many successful solution approaches use heuristics. One of the most prominent and widest applied problem for heuristics is the Vehicle Routing Problem (VRP). In the VRP we plan the visits of customers from a depot with vehicles in such an order that the transportation costs of the resulting delivery routes are as low as possible. Hereby, the capacity of the vehicles is limited so that usually more than one route is required. This problem is NP-hard and only small instances can be solved to optimality with linear solvers. To solve realistic-sized instances in a feasible time, a variety of heuristics have been successfully evolved in the last decades (Laporte, 2007).

A heuristic is comparable to human problem-solving in the way that it tries to solve a problem and employs changes whenever it gets stuck. These changes are usually stimulated by employing a certain degree of randomness. However, randomness is undirected and might not lead to promising solutions, like a mountaineer who is lost in a misty mountain range and randomly goes left and right. Thus, it might prove beneficial to guide a heuristic search with knowledge about the problem, just like giving the mountaineer a map.

Naturally this idea has been picked up in different heuristics. However, it has been used as a way of tuning a heuristic on-the-fly during the search process, to adapt it better to the problem instance at hand. For instance, adaptive large neighborhood search (Pisinger and Ropke, 2010) tries to find the best improvement operators during the solution process. In (Sörensen et al., 2008) the authors argue in favour of a similar self-adaptive concept within the family of variable neighborhood search. On the other hand, heuristics based on genetic concepts try to identify similarities that occur in a pool of good solutions, and use these similarities to construct better solutions (Baker and Ayechew, 2003). Likewise, some heuristics try to find and use 'backbones', i.e. solution parts that reoccur in different good solutions (Schneider, 2003).

However, rather than improving a heuristic on-the-fly for a particular instance, it would be much more valuable to already know beforehand how good solutions look like and how we find them.In the context of heuristics, this map is problem-specific knowledge, i.e. knowledge about the structure or characteristics of good solutions. That an analysis of the underlying problem is useful has already been demonstrated in the context of a scheduling problem (Aickelin and Dowsland, 2008). The authors of this study used empirical data to explore

the structure of a practical nurse scheduling problem, and incorporated the findings successfully in their heuristic. For the VRP, there has been some attempts to study and understand which heuristics work best for which kind of problem classes. This idea is picked up in the design of hyper-heuristics (Burke et al., 2003), which matches different types of problem or instance classes to different heuristics. However, these concepts focus once again on the understanding of the heuristic (should the mountaineer go left or right), and not on the problem itself (where is the highest peak).

In fact, we are not aware of any thorough analysis on problem-specific knowledge about the VRP. This can be explained by the complexity of the problem. It is very difficult to arrive at statements such as "intersections between two routes should always be eliminated" because the variety in problem instances and good solutions seems unmanageable.

However, with the advance of data mining there is a way out of this dilemma. Data mining techniques are powerful when it comes to revealing complex pattern and relations in huge amount of data that are hidden to human perception. Thus, the application of data mining to dig for knowledge about complex problems seems to be a promising step.

In this paper, we will conduct an exploratory study to investigate characteristics of good VRP solutions. In other words, we want to know what makes a solution good in comparison to worse solutions? We show how to generate data about a problem, and how to use this data to arrive at a deeper understanding about the problem structure itself. Finally, we demonstrate with an example that the incorporation of problem-specific knowledge can help to design more efficient heuristics. Even though we describe the whole process from the perspective of the VRP, we believe that the described techniques and ideas are applicable to a wide range of related and unrelated combinatorial optimization problems.

2. Data Mining and Knowledge Generation

The purpose of data mining is to find models that identify patterns and establish relationships in huge amounts of data. In contrast to statistics, the goal of these models is "not to infer the true distribution, but rather to predict future data as accurately as possible" (Shmueli, 2010). In other words, with data mining we aim to predict previously unseen data instead of explaining data at hand, the focus being on association rather than causation. We therefore also speak of predictive modeling. Even though predictive modelling can lack explanatory power, it is useful to find and test new theories in an exploratory fashion. Especially in

new and large datasets it is often difficult to hypothesize about complex relationships, and predictive modeling can help to unearth potential patterns which can serve as the basis for new hypothesis and theories. In this paper we use of data mining for theory generation.

The input to any predictive model is a data set. As a rule of thumb, with more available datapoints and richer information per datapoint, the predictive models become better. The information per datapoint is defined by its *features*. For instance, we could predict whether a certain client will be persuaded by a new marketing campaign with the help of data about a previous campaign. The dataset of the previous campaign could be composed of the features gender, age, family status and postal code, as well as the information whether the person responded to it.

We would feed this dataset to a *classification learner* which builds an internal model. In simple words, the model tries to find patterns in the data that distinguish those clients that responded to the previous campaign from those that did not. This process is also called *classification*. This internal model can then be used to predict future datapoints, i.e. whether potential addressees will respond to the new campaign. Nowadays, a huge variety of classification learners is available, from simple decision trees, to random forests (Breiman, 2001) and Support Vector Machines (SVM) (Hsu et al., 2003). The goal of the model is to maximise *predictive accuracy* or predictive power, which refer to the performance of the trained model on new data. In other words, the model tries to generalize patterns and relationships to classify new datapoints. The biggest obstacle to this *generalization* is overfitting the training data, i.e. the model is overly complex and too accurately fitting the training data. Therefore, the performance of a trained model is tested on so called validaiton sets, that were not included in the training set. Metrics to measure the performance are usually expressed by the number of correctly classified datapoints in the validation set.

We already mentioned that the predictive model does not necessarily offer causal explanation. However, it is possible to derive some causal theory. If we use simple linear models as predicitive models, explanation about the relative importance of different features can be made by looking at their coefficients. If the internal model is non-linear, we can use *rule extraction* to obtain a set of rules that explain the predictions (Martens et al., 2007). In this way, we are able to deduct theories from data. Since these theories are generalized findings that can be readily applied to new data, we will in the following use the term knowledge. In summary, data mining can be used to generate knowledge from large amounts of data in an exploratory fashion.

In this paper we distinguish two types of knowledge about combinatorial optimization problems. A problem is usually characterized by a set of relevant or representative instances, for instance benchmark instances. In order to be representative for the problem, those instances should be diverse with respect to relevant problem characteristics. In case of the VRP, problem instances can differ with respect to the number of customers or the position of the depot. *Class-specific knowledge* denotes findings that apply only to a certain class of those instances. For instance, a certain property might only hold for instances with relatively few customers. If, however, those findings can be generalized to all relevant instances, we speak of *problem-specific knowledge*. This knowledge is independent of properties of the specific instance. Both types of knowledge grant valuable insight about the nature of the considered problem. While problem-specific knowledge can be applied to all instances universally, class-specific knowledge allows to partition the set of instances according to distinguishing characteristics. In the following, we will demonstrate how to generate both types of knowledge with data mining.

3. From a problem to data

Our methodology is composed of three major steps. Firstly, we need to generate data, with which we build a predictive model, and finally we extract knowledge from the predictive model. While the latter two steps mainly involve technical challenges, the first one is highly exploratory and requires creativity and a focus on details. Since we can only deduct findings from the data that we create and the features we define, data generation is the most important and the most difficult step. In the following, we describe our data generation process for the VRP. Firstly, we generate random and diverse instances, for which we compute solutions, which we, finally, transform into a feature space.

3.1. Instance Generation

The VRP is a complex problem with a great variety of instances that can be varied according to the following attributes (Uchoa et al., 2014): (1) the positioning and number of customers, (2) the positioning of the depot, (3) the distribution of demand, and (4) the average route size or the number of routes, defined by the vehicle capacity. We try to sample a representative set of instances, by randomizing the choice of attributes for a particular instance within reasonable bounds. By the definition of these bounds, we automatically arrive at the definition of different instance classes.

Table 1: *Instance parameters for the different instance classes in the experiments*

Class	Number of Customers	Depot position	Demand	Routes
1	20-50	Center	[1,1]	3-6
2	20-50	Center	[1,10]	3-6
3	20-50	Edge	[1,1]	3-6
4	20-50	Edge	[1,10]	3-6
5	70-100	Center	[1,1]	6-10
6	70-100	Center	[1,10]	6-10
7	70-100	Edge	[1,1]	6-10
8	70-100	Edge	[1,10]	6-10

We define a set of small instances with 20-50 customers, and a set of bigger instances with 70-100 customers. Hereby, the customers are located randomly on a squared 1000x1000 grid. The clustering of customers is not considered in these experiments, since the clustered VRP is a special problem variant. We also define two classes for the position of the depot and the demand distribution, respectively. The depot is either located at the center of the grid, or at its edge. Likewise, the variance in customer demand is either rather high, or nonexistent, i.e. all customers have the same demand. For each combination of these three classes we generate one dataset and, thus, we generate eight datasets in total as presented in Table 1. Finally, we define the vehicle capacity randomly in such a way, that we have between 3 and 6 routes in the smaller classes and 6-10 routes in the bigger class. We remark that we could have introduced two more classes, one that contains instances with fewer and one with more routes. However, that would have increased the number of experiments beyond feasibility.

The separate analysis of different instance classes has two advantages. Firstly, differences in these datasets could point to class-specific knowledge, that only holds for certain instance characteristics. Reversely, if similar findings occur in all datasets we can interpret them as problem-specific-knowledge. Secondly, the instances within a dataset are expected to be more homogeneous since they share some attributes, which might enhance predictions on these datasets.

3.2. Computation of solutions

We want to understand what characterizes a good solution. Formulated differently, we can ask what distinguishes an optimal or near-optimal solution from a worse solution. If we find such distinguishing properties, they could help in

finding those good solutions more effectively, what is the goal of every heuristic. We investigate such distinctions and compute two solutions for each generated instance, one optimal and one non-optimal solution.

At the start, we experimented with obtaining the optimal solution with the solver CPLEX. However, CPLEX might need several minutes to compute a solution even for a smaller instance. To conduct our experiments in a feasible time, we thus use a heuristic to compute the optimal solutions. The heuristic is based on guided local search and described briefly in Section 5, without incorporating any knowledge. We refer the interested reader to (Arnold and Sörensen, 2016) for a more elaborate description. For the considered problems it finds the optimal solution in 90% of the cases, and otherwise the costs are only about 0.1% higher. Thus, the computed solutions are mostly optimal, or very close to being optimal, and in the following we denote them with 'optimal solution'.

Next to the optimal solution, we also compute a non-optimal solution for each instance. We hereby define non-optimality in the sense that the respective solution value has a certain gap to the optimal value. In the experiments we use two different classes of gaps. In the first class, the non-optimal solutions have on average a gap of 4% (the individual gap can range from 3% to 5%). Solutions with such a gap are not particularly good, and can be obtained with relatively simple construction heuristics. In the second class, the non-optimal solutions are better, having an average gap of 2% (the individual gap can range from 1% to 3%). We use two different classes of non-optimality to investigate if and in how far the degree of non-optimality impacts differences in the solutions. Intuitively, better solutions should exhibit less structural differences to optimal ones.

The computation of non-optimal solutions has to be executed with care. Whereas there might be only a few or even only a unique optimal solution for an instance, there are many more solutions that exhibit a similar gap. In general, with a larger gap there are more corresponding solutions. Therefore, the structure and characteristics of the non-optimal solution might be biased towards the solution process. For instance, if the solution process focuses on removing long edges, the computed solutions will differ from those that are computed by focusing on intra-route optimisation. We addressed this possible issue by utilizing two different heuristics H1 and H2 to generate non-optimal solutions. Heuristic H1 is the same that we use in the computation of the optimal solution, with the difference that it aborts the search early if a solution with the aspired solution value has been found. On contrary, heuristic H2 is a constructive heuristic based on the Clark and Wright heuristic **??**. Thus, we generate non-optimal solutions with two completely different approaches.

3.3. Metrics for a solution

To discover characteristics that are typical for good solutions, we need to 'measure' a solution with defined metrics. With measuring we here mean transforming the structure of a solution into quantitative metrics, which later serve as input to the predictive model. This step is highly exploratory, since there are no guidelines about which metrics should be included. As a rule of thumb, the more metrics we can define, the better the chance that the classification learner unearth patterns. We used the following metrics to characterize a computed solution:

S1 - Number of intersections between edges

S2 - Longest distance between two connected customers, per route

S3 - Average distance between depot to directly-connected customers

S4 - Average distance between routes

S5 - Average width per route

S6 - Average span in radian per route

S7 - Average compactness per route, measured by width

S8 - Average compactness per route, measured by radian

S9 - Average depth per route

S10 - Variance in number of customers per route

These metrics and their measurement are visualized in Figure 1. The first two metrics reflect observations in the field of Operations Research, especially with regard to the famous Traveling Salesman Problem. We expect that good solutions tend to have fewer intersections and less extremely long edges. In comparison, the impact of the other metrics seems less well-established. Metric S3 is clearly exploratory, and we hypothesize that in good solutions, edges connecting the depot and customers should be rather short. Intuitively, routes should be clearly separated (S4) and not overlap too much, both in width (S5) and in depth (S9). Thus, we expect good solutions to have less wide and deeper routes on average, resulting in a relatively high distance between them. The width of a route can also be interpreted by looking at the customers' radians toward the depot (S6).

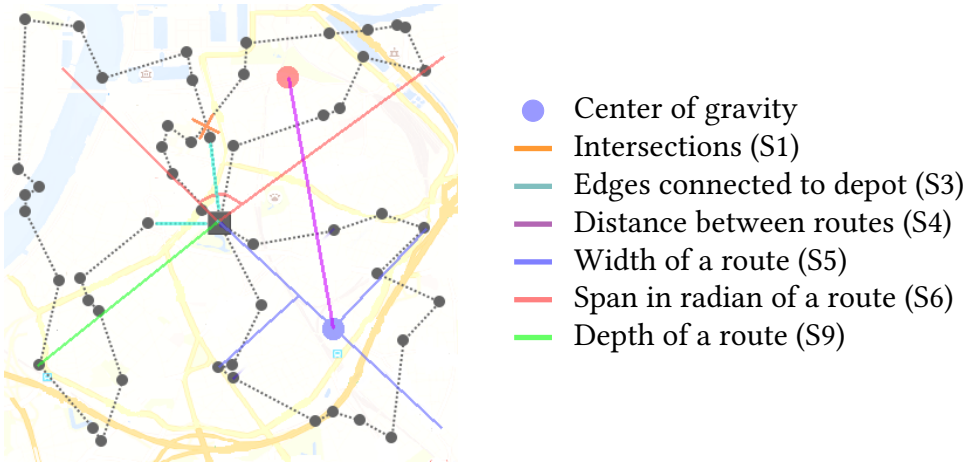
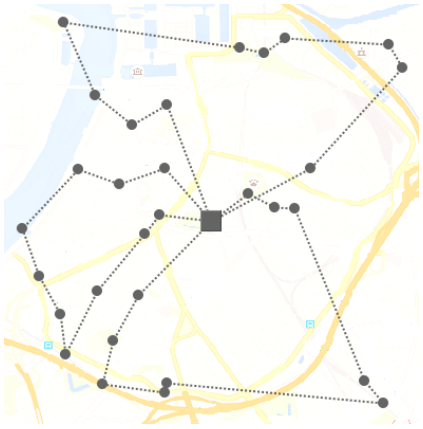


Figure 1: *Metrics to obtain characteristics of a solution.*

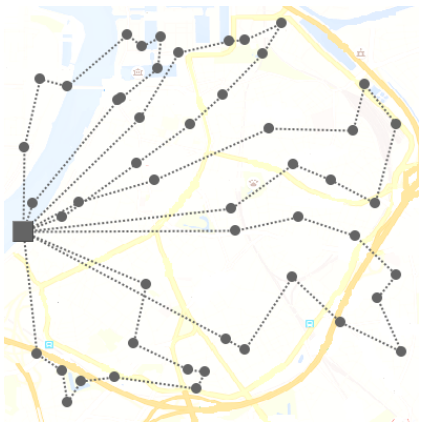
Narrow routes also tend to have a smaller difference in the radians of its customers, which is used in the popular sweep heuristic to construct routes. Routes can be shaped more like a line (very compact) or more circular (less compact), which we measure with S7. Note that the metric compactness is quite similar to the metric width, as compact routes tend to be less wide. Then, compactness can also be measured by the variation in radians of a route's customers (S8). Finally, with S10 we measure how balanced the routes are in terms of the number of delivered customers.

More formally, the metrics are computed as follows. We obtain the distance between routes r_1 and r_2 (metric S4) by computing the distances between the respective centers of gravity g_{r_1} and g_{r_2}. The center of gravity of a route is defined as the average of all coordinates of its nodes, including the depot. This coordinate is also used in the computation of the route width (S5). For each customer on route r, we derive its distance from the line that connects the depot and g_r. The width of route r is then defined as the sum of the distances between this line and those two customers that are furthest away from this line on either side (thus, by definition a route with only one customer has always a width of 0). Likewise, the compactness (S7) is derived by computing the average distance between the route's customers and the line between the depot and g_r. The depth of a route (S9) is simply defined as the largest distance between a customer on the route and the depot.

We remark that the number of routes is usually also part of the solution in the standard VRP. However, in this study we assume that the number of routes



Average width (optimal): 521
Average width (non-optimal): 540



Average width (optimal): 205
Average width (non-optimal): 230

Figure 2: *The width (and all other solution metrics) can vary between different instances.*

is defined beforehand. The reason for this assumption is that we investigate the impact of the structure of a solution on its quality, and a different number of routes might change the solution structure significantly.

3.4. Metrics for an instance

The values of the above solution metrics are dependent on the respective instance. Consider Figure 2 for an example. For both instances, the average width per route is lower in the optimal solution, from which one could conclude that low width is a predictor for good solutions. However, the values are not comparable between the two instances, since the instances vary in the number of routes and the location of the depot location. The classification learner does not know whether two sets of solution metrics (for the optimal and the non-optimal solution) stem from the same instance and are paired, and only comparing the absolute values will lead to wrong or no predictions. Thus, we need to normalize the solution metrics with respect to the respective instance. This requires the definition of, possibly all, relevant metrics that capture the characteristics of an instance. We utilized the following metrics:

I1 - Number of customers

I2 - Number of routes

I3 - Degree of capacity utilisation

I4 - Average distance between each pair of customers

I5 - Variance in the pairwise distance between customers

I6 - Average distance from customers to the depot

I7 - Variance in the distance from customers to the depot

I8 - Variation in the radians of customers towards the depot

Metrics I1 and I2 are basic instance parameters. The degree of capacity utilisation (I3) is given by the sum of demand divided by the available capacity in all vehicles. This is value is the same for all computed solutions per instance, since we fix the number of vehicles. The last five metrics indicate the relative position of customers to each other and to the depot. For instance, I4 is lower if the customers are more clustered, and higher if they are uniformly spreaded. Likewise, I7 is lower if the depot is in a more central position. For each customer we also determine the radian from the perspective of the depot. Then, I8 has a low value, if the depot is located at the fringe and the customers are more clustered.

We achieve normalization with respect to these instance metrics within the data mining framework. The instance metrics form, along with the solution metrics, the input of the classification learner. Then, the interdependencies between solution and instance metrics are learnt. We want to remark that this approach might require larger datasets since it comes at the cost of more features and more interdependencies need to be learnt.

4. From data to problem-specific knowledge

On the basis of the previous description we generate 32 datasets. For each of the eight instance classes we vary the generation of the non-optimal solution with respect to the gap (either 2% or 4%) and the heuristic (either H1 or H2). For the smaller instance classes (20-50 customers) we generate 5.000 instances per dataset. Larger instances take longer to solve and, thus, for the instance classes with 70-100 customers we limit the data to 1.000 instances. For each generated instance we compute an optimal and a non-optimal solution. Altogether, we computed 192.000 VRP solutions in the course of this study. For each computed solution we derive the corresponding solution metrics S1-S10 and instance metrics I1-I8, which, together with a flag for whether the solution is optimal or not, constitute one data-point of the dataset.

Table 2: *Prediction accuracy with linear SVM for each dataset*

			2% gap		4% gap	
		#Datapoints	H1	H2	H1	H2
20–50 cust.	Class 1	10.000	66%	62%	72%	64%
	Class 2	10.000	64%	61%	75%	63%
	Class 3	10.000	67%	68%	76%	75%
	Class 4	10.000	66%	65%	74%	71%
50–80 cust.	Class 5	2.000	81%	81%	91%	90%
	Class 6	2.000	82%	81%	90%	89%
	Class 7	2.000	86%	86%	90%	93%
	Class 8	2.000	81%	82%	89%	89%

We then analyse each dataset with a classification learner in Matlab, using 5-fold cross validation. In short, the learner uses a certain percentage of randomly-selected datapoints (training set) to train an internal predictive model which tries to distinguish between optimal and non-optimal solution. This predictive model is then validated with the help of the remaining datapoints (validation set). The percentage of correctly classified datapoints in the validation set reflects the prediction accuracy of the predictive model. Since the number of datapoints associated with optimal solutions is the same as the one for non-optimal solutions, a prediction accuracy of higher than 50% reveals that there are predictive pattern within the data.

4.1. Prediction accuracy

As classification learner we use Simple Trees, Random Forests and linear SVMs. Hereby, SVMs performed best for all datasets, and in Table 2 we report the corresponding prediction accuracies. From these result we can make the following observations.

For all datasets we obtain prediction rates that are significantly higher than 50%. This predictability in solution structure depends on both, the gap and the instance class, and reaches up to 93%. In other words, if you give our predictive models any solution for any instance (within the defined parameter bounds), it will, with an above random-chance probability between 61% and 93% tell you whether the solution is optimal (or very close to optimal) or not. This is a remarkable finding in itself, since it reveals that there is predictive power in at

least some of the solution metrics we defined above. Thus, some values for the characteristics S1 to S10 are, alone or in interplay with others, more characteristic for an optimal solution than for non-optimal ones.

Secondly, the way we generate the non-optimal solution, either by H1 or H2, does not seem to affect the prediction accuracy much. We only observe differences for smaller instances with a centric depot location (Class 1 and Class 2), for which the learner seems to struggle with finding distinctive characteristics of non-optimal solutions generated with H2.

Thirdly, a smaller gap between the non-optimal solution and the optimal solution results in a lower prediction accuracy. This was to be expected, since, with a lower difference in the solution value, the two solutions are more likely to be also more similar in structure. This similarity makes it more difficult to find distinguishing features, which in turn decrease prediction accuracy.

Finally, problems with a larger amount of customers have a higher prediction accuracy, i.e. classes 5-8 have a higher accuracy than their counterparts 1-4. Since these problems exhibit a higher complexity with more customers connected on more routes, they are also richer in information. Thus, it is easier for the classification learner to identify patterns and relationships. Similarly but much less strongly, the predictability seems to be slightly higher for problem instances where the depot is located at an edge (e.g. Class 3 versus Class 1). On the other hand, there are no systematic differences between instance classes that vary in the distribution of customer demand (e.g. Class 5 versus Class 6). This suggests that the effect of these two parameters on structural differences in routing solutions is less impactful than the number of customers.

In general, we can identify four distinctive families which are similar in terms of prediction accuracy, and thus, in terms of structural differences between optimal and non-optimal solutions. Classes 1-4 have a predictive accuracy around 65% for a 2% gap, and around 73% for a 4% gap. Meanwhile, the accuracy for classes 5-8 is around 83% and 90%, respectively.

4.2. Explaining prediction

These results raise the question 'What are predictable differences between solutions of different quality?'. To obtain some explanations for why a classification learner predicted the way it did, we can estimate the individual contribution of each metric to the prediction model.

Thereby, in each dataset, we only consider one solution metric and the instance metrics for normalization, and use a simple tree to build a predictive model. The prediction accuracy of this model then reflects the predictive power

Table 3: *Solution metrics with the highest individual prediction accuracy per instance class. I - Intersections (S1), F - First edges (S3), W - Width (S5), R - Radians (S6), C - Compactness (S7 and S8). The numbers present the prediction accuracy.*

	2% gap					4% gap				
	I	F	W	R	C	I	F	W	R	C
Class 1					**58**	57			59	**61**
Class 2					**58**	57			**62**	59
Class 3	58		**60**		59	61		**65**	60	64
Class 4	57		**58**		58	59		**62**		62
Class 5		63	67	**68**	67	60	71	**79**	77	78
Class 6		61	65	66	**69**	60	68	**74**	72	74
Class 7	67	61	**79**	65	75	71	65	**84**	72	80
Class 8	64		**72**	62	70	68		**79**	68	77

of the respective solution metric. We found that the corresponding results are similar for both heuristics H1 and H2, and thus, in the following we report the results as the average over both heuristics. In other words, H1 and H2 produce solutions that are similar in structure.

Table 3 highlights for each class the most predictive individual solution metrics and their prediction accuracy if they are higher than 56%. From these results we can observe the following findings.

Some of our solution metrics explain most of the prediction accuracy, while others have no or only little predictive power on their own. Surprisingly, longest edges that are not connected to a depot (S2) have only little predictive value. Likewise, the distance between routes (S4), their depths (S9) and the variance in the number of customers per route (S10) do not seem to differ too much between optimal and non-optimal solutions.

On contrary, the length of edges that are directly connected to the depot (S3) are a more distinguishing characteristic. The shorter they are, the better the solution tends to be. Similarly, better solution have fewer intersection (S1) and can be classified as such with up to 71% accuracy. However, the most predictive metrics in distinguishing optimal from non-optimal solutions are those associated with wideness, radiants and compactness of routes (S5-S8). In general and for almost all instance classes except for Class 1 and Class 2, the wideness (S5) seems to be the best indicator of whether a solution is optimal or not. By only considering

Table 4: *Extracted rules from the classification tree for the dataset of class 6 with a 2% gap*

Solution Metrics				Instance metrics		
S1	S3	S6	S8	I4	I8	
	> 167		> 0.23			$\rightarrow$ Non-optimal
	> 199	> 0.97	> 0.27		< 1.6	$\rightarrow$ Non-optimal
> 0.26	> 222	> 0.73	> 0.27	> 152		$\rightarrow$ Non-optimal
	> 240	> 0.74	> 0.23			$\rightarrow$ Non-optimal
	> 294		< 0.23			$\rightarrow$ Non-optimal
...	...				...	

width of routes we can correctly classify a non-optimal solution with up to 84% accuracy. But also the compactness and radiants, which measure similar structural elements, are highly predictive. In summary, the four most characteristic properties of optimal VRP solutions are narrow, compact routes with few intersections and short outgoing edges from the depot. These findings hold for all instance classes, and can therefore be considered as problem-specific knowledge.

We also observe some differences between the classes. The radiants seems to be a slightly better metric for instances where the depot is in the center. A similar observation holds for the depth of routes, even though the predictive accuracies are not high. On contrary, if the depot is located at an edge, the distinction between optimal and non-optimal solutions is greater with respect to the width and the number of intersections. On the other hand, the demand distribution does not appear to have any significant effect on differences in route structure.

Another interesting observation is that the relative prediction strength of metrics does not change with different gaps. Even though the absolute prediction accuracy is lower for 2% gaps compared to 4% gaps, the ranking of most predictive metrics is almost the same. A possible explanation is that these metrics are predictable in most cases, independent of the gap. Another explanation could be that the considered gaps are too close to each other to observe significant differences.

4.3. From explanations to rules

So far, we only considered solution metrics in isolation to explain results. In the following, we also want to consider the interaction between metrics and arrive at rules that tell us, if and why a certain solution is classified as non-optimal. For each dataset we build a simple classification tree. A classification

tree branches according to the most predictive metrics, and its leaves contain the class label ‘optimal’ or ‘non-optimal’ . The branching towards each leaf can be transformed into a rule, represented by a set of value ranges for certain features. If a solution fulfills these conditions, it is classified according to the defined class label. Table 4 displays extracted rules for instance class 6 with a 2% gap.

These rules constitute more detailed knowledge about the differences between optimal and non-optimal solutions, and might be of great value in practice. Imagine we had a non-optimal solution for a certain instance, and we would like to know, how this solution could be further improved. We simply compute the respective solution and instance metrics, and find the rule, or rules, that classifies the solution as non-optimal (if the solution is already optimal or close to optimal, there might not be such a rule). The conditions of this rule then indicate likely reasons for why this solution is non-optimal.

As an example, consider the first row in Table 4. If this rule applies, our solution is likely to be non-optimal because the outgoing edges from the depot are too long and the routes are not compact enough. With this information we can guide the search process into a promising direction. In the case of a local search, we could select operators that tend to make routes more compact and, thereby, focus especially on the edges connected to the depot. In the case that the routes are sufficiently compact, but the edges to the depot are too long (fifth rule in Table 4), we can take even more targeted action and focus specifically on those edges. Reversely, it might be difficult to infer any meaningful actions if the rules are too complex. For instance, the third rule applies, if our solution has too many intersecting edges while the routes are too wide and non-compact, and edges to the depot are too long. There is probably no operators that can dealt with all of these issues at once.

Since the rules hold for many solutions and different instances, they can also be used to develop new operators. With regard to the fifth rule, we could try to build a local search operator which exchanges customers that are connected to the depot while keeping the remaining parts of the routes fixed.

We do not argue that this process works for any instance or at any stage of the search, nor that rules like ‘make routes more compact’ can be readily translated into actions. However, in general it has proven to be the most distinguishing property over a large sets of solutions, given the respective instance and solution. At this point, it also becomes clear how far we can go when we speak of knowledge: It is not a provable if-then cooking recipe, but rather *a set of classification rules that hold with an above-random-chance probability for similar types of instances.* Due to the complexity of the VRP, this might present the most granular

form of knowledge that can be found.

5. Application of problem-specific knowledge in the design of heuristics

In the sections above we have derived insights into the nature of good VRP solutions. It would be even better if they could help in building better heuristics in practice. In the following, we demonstrate with the help of a case study how problem-specific knowledge could be used as guidance in the search process and, thereby, how knowledge-generation can support heuristics.

The search of a heuristic can be guided in different ways. If the heuristic involves a variety of operators, we can influence and adapt the choice as to which operator should be used when and how often. For instance, in adaptive large neighborhood search we have to choose among many local search operators, while genetic algorithms usually offer a choice between different crossover operators. On the other hand, we could also choose where those operators should be used, and thereby concentrate the search on a certain aspect or area of the current solution. In this example we will use problem-specific knowledge to improve the latter, i.e. to focus the search effort on promising areas of a solution. More precisely, we try to find those edges that should be removed from the current solution by means of penalisation.

The idea behind the penalisation of edges, or solution features in general, is based on Guided Local Search (GLS) (Voudouris and Tsang, 2003). GLS has already been successfully applied to the TSP (Voudouris and Tsang, 2003) and the VRP (Mester and Bräysy, 2007) and stands out due to its simplicity and effectiveness. Its idea is to penalize a 'bad' edge by changing the cost $c(i, j)$ of that edge between customers i and j to c^g:

$$c^g(i, j) = c(i, j) + \lambda p_{(i,j)} c(i, j), \tag{1}$$

where $p_{(i,j)}$ counts the number of penalties of edge (i, j) during the search, and λ (usually chosen to be in $[0.1, 0.3]$) controls the impact of penalties. After the penalisation of an edge, we try to remove this edge by means of a local search. Since the local search only focuses on a certain area of a solution, i.e. the penalized edge, it can be executed very efficiently, and thus, the two steps of edge penalization and local search can be iterated thousands of times.

Naturally, the effectiveness of this heuristic approach depends on the ability to detect (and remove) 'bad' edges. One of the most efficient heuristics to date penalizes the longest edges (Mester and Bräysy, 2007), a straight-forward idea

since longer edges contribute more weight to the objective function. Indeed, our study confirmed that longer edges, especially those connected to the depot, have a certain predictive power to classify good solutions. Even more important, we showed that a low width and high compactness of routes as well as few intersections are a distinguishing feature of good solutions. It seems to be a logical step to use this knowledge in defining a function $b(i,j)$ that measures the 'badness' of an edge (i,j). We try to reduce the width and increase compactness of routes by penalizing edges that are wide, as measured by metric S7. Additionally, we count the number of intersections of each edge, and arrive at the following function:

$$b^k(i,j) = \frac{\Big(c(i,j) + w(i,j)\Big)\Big(1 + 0.1I(i,j)\Big)}{1 + p_{(i,j)}}, \tag{2}$$

where $c(i,j)$ indicates the original cost value or length of edge (i,j), $w(i,j)$ denotes the width and $I(i,j)$ counts the number of intersections with other edges. In this way, the edge in the current solution with the highest function value is penalized according to (1), and the succeeding local search tries to remove it. The denominator makes sure that we do not penalize the same edge over and over, especially when it reappears in a solution, but rather spread penalisation among the entire search space.

We demonstrate that this simple extension of an already powerful heuristic framework with problem-specific knowledge enhances the framework's efficiency even further. Our methodological approach is the following. We use the popular benchmark instances of Augerat and Golden to compare the effectiveness of a GLS with traditional edge penalisation b^t and a GLS with knowledge-driven penalisation b^k. The instances by Augerat are relatively small with 32 to 80 customers, while the instances by Golden include 200 up to 488 customers. As stated above, the traditional GLS penalizes the longest edges, which have the highest value in the function

$$b^t(i,j) = \frac{c(i,j)}{1 + p_{(i,j)}}. \tag{3}$$

After an edge has been penalized, several powerful local search operators try to remove it from the current solution. We use the ejection chain, the CROSS-exchange and the n-opt operator to conduct the local search. One iteration is then defined by a sequence of edge penalisation and local search. For more details about these operators as well as in-depth description and analysis of the heuristic we refer the interested reader to (Arnold and Sörensen, 2016).

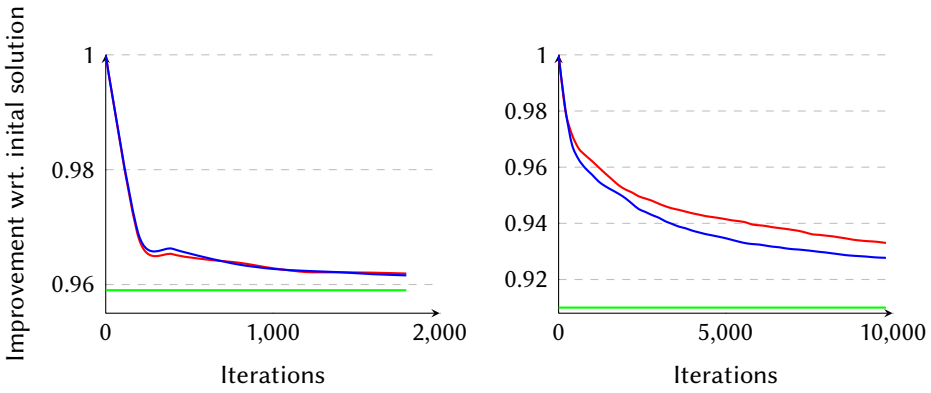


Figure 3: *Average Improvement after an increasing number of iterations for 20 smaller instances by Augerat (left) and 12 larger instances by Golden (right). The blue line represents knowledge-based edge penalisation with (2), while the red line represents the traditional edge penalisation according to (3). The green line reflects the optimal solutions.*

The graphs in Figure 3 visualize for both GLS versions the average improvement with an increasing number of iterations over all instances in the two benchmark sets. The improvement is given with respect to the initial solution computed with the Clark-Wright heuristic. Since the only difference between both GLS versions is the definition of the 'badness' function, both heuristics are similar in computation time if metrics $I(i, j)$ and $w(i, j)$ can be measured efficiently. Indeed, by means of preprocessing and updating, this can be achieved. We compute and memorize the width and number of intersections for each edge in the first solution, and only update these values for edges that change. Thus, both heuristics require the same computation time, and yet, we can observe that the knowledge-driven GLS version seems to perform better. While the differences in the set of smaller instances are very small, in the set of larger instances the knowledge-based version clearly outperforms the traditional version at every stage of the search.

These findings suggests that problem-specific knowledge becomes more valuable when problem instances grow in complexity. With a higher complexity, the number of possible options at each step of the search grows as well, and thus, even some knowledge can help to take better decisions. Analogously, a mountaineer will find the highest peak even without a map, if the mountain area is small. The larger the area is, the more helpful becomes a map.

6. Discussion

In this work we introduced a framework to derive problem-specific knowledge for combinatorial optimization problems. The core of the framework is data mining, a technique to find and extract pattern and generalize it to new data. Hereby, the most challenging step is the definition of relevant metrics to measure the structure of a solution and that of an instance. This is a highly exploratory task which requires a deeper understanding about the problem as well as some creativity.

Furthermore, the framework requires a random-instance generator as well as solution techniques to generate solutions of varying quality. Since we need to generate a huge amount of datapoints for the classification learner to work, the solution technique should be efficient and, in the best case, be able to produce almost optimal solutions. This requirement might present a high hindrance when applying this technique to other combinatorial problems, for which efficient solution methods are not yet available. Reversely, one might argue that if we already have an efficient algorithm to find optimal solutions for a problem, like in the case of the VRP, why would we need further insights into the problem structure?

To address these questions, it might not always be necessary to compute optimal solutions, but one might also obtain valuable findings by comparing good with worse solutions. Additionally, if the solution techniques are not efficient, we can limit the data to small instances. There is no guarantee that the findings can then be generalized to bigger instances, but such an analysis might still reveal some understanding about the problem, especially when the problem is new and one wants to explore its solution structure. On the other hand, even if we already have efficient techniques to tackle a problem, we demonstrated that even for a widely popular problem as the VRP we are able to find simple characteristics that explain what makes a solution good. More precisely, we found that the compactness, the width and the span in radians of routes, as well as the number of intersecting edges and the distances of connecting edges to the depot are highly distinguishing features between optimal and non-optimal solutions.

Interestingly, those findings hold for a variety of instance classes which suggests that structural differences in optimal and non-optimal solutions are similar across a wide range of instances. From this observation we can draw the implication (or validation for what is already widely assumed) that the same solution technique will behave similar for a wide set of instances. In other words, if a heuristic performs well on a representative set of benchmark instances, it is

likely to perform similarly well on other sets.

Finally, we showed that these characteristics can be used to make good heuristics even better by guiding the search process. In how far problem-specific-knowledge can be incorporated in a particular heuristic, might depend on its design. In our case, we simply adapted the evaluation function for edges to obtain significant better results. In general, it seems reasonable to build a heuristic around problem-specific knowledge, rather than the other way around. If we know how good solutions look like, it should be straight-forward to design operators and functions to lead us there.

In this work we only scratched at the surface of what might be possible. With the help of the deducted set of rules, the guidance could be designed in a much finer case-by-case fashion. In each step of the heuristic we could identify relevant rules and adapt parameters and operators accordingly, e.g. the prioritisation of the elimination of certain intersections over the reduction of route width. This process could be the first step in the automation of heuristic design. Given a particular problem together with a database of metaheuristic designs, operators and problem-specific knowledge, an efficient heuristic could be developed almost autonomously, complemented with the creativity and experience of developers.

Uwe Aickelin and Kathryn Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *arXiv preprint arXiv:0802.2001*, 2008.

F. Arnold and K. Sörensen. A knowledge based local search heuristic for routing problems. *Working paper, University of Antwerp*, 1(1):1, 2016.

Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

Gilbert Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics (NRL)*, 54(8):811–819, 2007.

David Martens, Bart Baesens, Tony Van Gestel, and Jan Vanthienen. Comprehensible credit scoring models using rule extraction from support vector machines. *European journal of operational research*, 183(3):1466–1476, 2007.

David Mester and Olli Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34 (10):2964–2975, 2007.

David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

Johannes Schneider. Searching for backbonesfi!?a high-performance parallel algorithm for solving combinatorial optimization problems. *Future Generation Computer Systems*, 19(1):121–131, 2003.

Galit Shmueli. To explain or to predict? *Statistical science*, pages 289–310, 2010.

Kenneth Sörensen, Marc Sevaux, and Patrick Schittekat. fimultiple neighbourhoodfi search in commercial vrp packages: Evolving towards self-adaptive methods. In *Adaptive and multilevel metaheuristics*, pages 239–253. Springer, 2008.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Anand Subramanian, and Thibaut Vidal. New benchmark instances for the capacitated vehicle routing problem. Technical report, Research Report Engenharia de Produção, Universidade Federal Fluminense, 2014.

Christos Voudouris and Edward PK Tsang. *Guided local search.* Springer, 2003.