A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction

Jana Ries[1*], Patrick Beullens[2]
[1]Portsmouth Business School, University of Portsmouth,
[2]School of Mathematics, School of Management, University of Southampton,
jana.ries@port.ac.uk, p.beullens@soton.ac.uk

Abstract

Two main concepts are established in the literature for the Parameter Setting Problem (PSP) of metaheuristics: Parameter Tuning Strategies (PTS) and Parameter Control Strategies (PCS). While PTS result in a fixed parameter setting for a set of problem instances, PCS are incorporated into the metaheuristic and adapt parameter values according to instance-specific performance feedback.

The idea of Instance-specific Parameter Tuning Strategies (IPTS) is aiming to combine advantages of both tuning and control strategies by enabling the adoption of parameter values tailored to instance-specific characteristics a priori to running the metaheuristic. This requires, however, a significant knowledge about the impact of instance-characteristics on heuristic performance.

This paper presents an approach that semi-automatically designs the fuzzy logic rule base to obtain instance-specific parameter values by means of decision trees. This enables the user to automate the process of converting insights about instance-specific information and its impact on heuristic performance into a fuzzy rule base IPTS system. The system incorporates the decision maker's preference about the trade-off between computational time and solution quality.

Keywords: Heuristics, Parameter calibration, Fuzzy systems, Decision trees, Travelling Salesman Problem.

*Corresponding author

1 Introduction

The Parameter Setting Problem (PSP) is the search for a set of algorithm-specific parameter values to improve metaheuristic performance. Solving the PSP is important for ensuring the efficient implementation of a metaheuristic approach to combinatorial optimisation problems (Hooker 1995, Johnson 2002).

Several methodologies for the PSP have emerged during the last decade, leading to the main strategies of: Parameter Tuning or PTS (Coy *et al.* 2001, Adenso-Diaz & Laguna 2006, Kern 2006) and Parameter Control or PCS (Battiti 1996, Eiben *et al.* 1999, Jeong *et al.* 2009), which are also referred to as offline and online approaches, respectively. PTS look for one fixed set of parameter values to be used for each instance to be solved later. The determination of this best set of values occurs a priori by an examination of metaheuristic performance on a representative set of instances. PCS dynamically adjust the parameter values to instance-specific values during the metaheuristic search on the instance.

A recent addition are the so-called Instance-specific Parameter Tuning Strategies (IPTS). IPTS aim to combine advantages of PTS and PCS, and to incorporate an explicit knowledge of the impact of instance characteristics and decision maker preferences about the trade-off between solution quality and computational time. As in PTS, a representative set of instances is first investigated. Rather than aiming to obtain one specific 'robust' set of parameter values, the instance set is used to design an efficient tuning method that can return instance-specific parameter values based on measurable instance characteristics. Prior to running the metaheuristic on a particular new instance, the instance is examined by the tuning method and appropriate parameter values are returned to initialise the metaheuristic, which is then applied to the instance. Only few approaches can be currently found in the literature that consider the importance of instance-specific information. Existing ones use case-based bayesian reasoning (Pavon *et al.* 2009) or fuzzy logic (Ries *et al.* 2012), while Kadioglu *et al.* (2010) propose a clustering approach with parameter tuning systems that

ignore structural information of instances, i.e. ParamILS (Hutter *et al.*, 2007) and Gender-Based Genetic Algorithm (GGA) (Gil *et al.*, 2009). Hence, the authors cluster similarly structured instances and subsequently associate the new instance with the parameter setting that has been found using a non-instance-specific tuning tool.

Which of these methods is best is difficult to assess in general. PTS do not require the adaptation of the metaheuristic, nor an explicit knowledge of instance characteristics, nor an explicit knowledge of the impact of parameter values on heuristic performance. Once the set of robust parameter values is determined from a representative set of instances, they are simple to use. However, as the set of parameter values is typically chosen as to maximise the average performance over a set of instances, the parameter values are typically not optimal for any particular instance of the set. In other words, the approach can be expected to perform well as long as the metaheuristic appears to behave robustly and 'is relatively insensitive to differences in problem characteristics, data quality and parameter tuning' (Barr *et al.* 1995), at least for the area of application from which the instances are drawn.

PCS often need the modification of the metaheuristic algorithm, or at least allow for dynamic communication of this algorithm with an external parameter control procedure. Some knowledge of how algorithm-specific parameter values influence heuristic performance is required in order to adjust the parameters dynamically. Instance-specific information, however, is used only implicitly through an observed heuristic performance in an iterative process of adjusting, for example, intensification and diversification efforts of the search routine. PCS work arguably well when the instances that need solving are not a priori well known but are expected to differ significantly in their characteristics so that dynamically tailoring the parameter values pays off in solution quality or computational time.

Several experimental studies have outlined that the investigation of structural information implicitly present in a problem instance, and its impact on heuristic performance, can be crucial in improving parameter setting methods (Coy *et al.* 2001, Johnson 2002, Kern 2006, Ridge 2007). This supports the idea of having a clear advantage from using a well structured

approach to calibrating a metaheuristic based on instance-specific information.

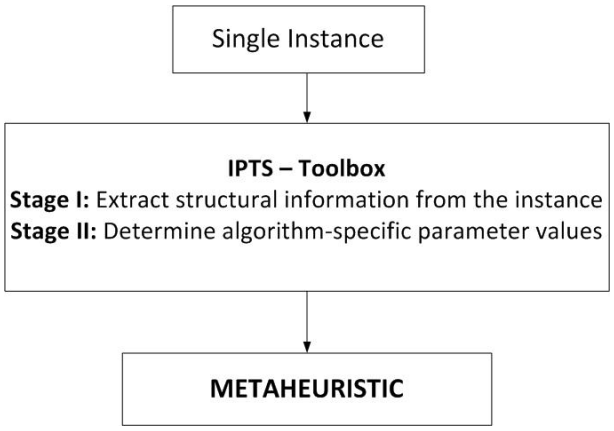

Figure 1: IPTS in the use phase

In IPTS, the interaction between some measurable instance characteristics and algorithm-specific parameter values is to be explicitly incorporated in the design of the tuning strategy. A representative set of instances is used to design the tuning method. Once designed, the tuning method calculates the characteristics of any given instance, and then applies a method to return a set of instance-specific parameter values. These values are subsequently used to initialise the metaheuristic in order to solve the particular instance, see Figure 1. The approach therefore aims to preserve much of the simplicity of PTS but allows for the use, like in PCS, of instance-specific parameter values. This set-up aims to avoid the additional computational efforts during the metaheuristic search required in PCS, and designers of IPTS can use any metaheuristic of which the code is not explicitly available; it may e.g. only be available as a callable routine with the ability to set its parameter values.

In a further attempt to facilitate and adjust a parameter setting strategy for practitioners, a decision maker preference parameter p can be easily introduced in IPTS. This parameter can be interpreted as the 'time pressure' under which a solution needs to be obtained. The parameter takes values between 0 and 1, the former value indicating no time pressure and the latter the need for short computational times. Any in-between value describes a preference as a weighted combination of both performance values, which ideally should correspond with

parameter settings that follow the pareto front in the two-dimensional objective space of solution quality and computational time. From the point of view of the practioner, the IPTS replaces the problem of choosing a set of metaheuristic parameter values by the problem of choosing an appropriate value for p.

An IPTS design poses several challenges including the selection of relevant characteristics and the identification of the relation between the instance-specific information, the parameter values of the algorithm, and their impact on running times and quality of solutions obtained. The current IPTS in the literature provide the structure for a successful calibration concept. The design of this calibration system itself, however, is based on expert judgement and a manual set-up, irrespective of whether further learning during the search is either incorporated (Pavon *et al.* 2009) or not (Ries *et al.* 2012). The design of the tuning method in the latter study, for example, is based on an extensive statistical analysis for identifying main and interaction effects between instance characteristics, algorithmic parameters of the metaheuristic, and performance of the metaheuristic. Subsequently, it requires an expert to interpret the statistical results in order to manually specify a meaningful set of fuzzy decision rules.

In order to overcome these difficulties, this paper suggests a semi-automated approach for designing (fuzzy logic) IPTS whereby the classification rules are derived from automatically generated decision trees. Hence, the knowledge of the interaction between instance- and algorithm-specific parameters does not need to become explicit to the designer, but will be implicitly and automatically transferred into a (fuzzy) rule base for parameter tuning. The proposed approach of rule induction from induced decision trees is quite generally applicable for developing an IPTS for any metaheuristic and type of problem. To make the application concrete in this paper, the approach is used for the development of a specific tuning method of the class of fuzzy logic-based IPTS.

The remainder of the paper is organised as follows. Section 2 presents the test cases for the symmetric and asymmetric Travelling Salesman Problem (TSP) in combination with

Guided Local Search (GLS). The outline of the design of an automated rule base for a fuzzy system using decision trees is found in Section 3, and results are outlined in Section 4. Section 5 gives conclusions and recommendations for future research.

2 Description of the test case environment

The main aim of this article is to outline and demonstrate the potential of a *semi-automated approach* to design a fuzzy tuning method for IPTS. Its value is hence foremost to be determined from a comparison with the approach in which expert judgement and a manual set-up is used, as in Ries *et al.* (2012). Arguably, the latter approach is time consuming for the programmer as it requires a careful explicit interpretation and translation of the statistical results into a fuzzy rule base. The automated approach avoids that any statistical tests need to be conducted, interpreted, and translated into a rule base, and hence will produce the tuning rules in principle much quicker. The question is hence whether the automated approach gives a tuning method of at least comparable average performance. To facilitate this comparison, we hence use the same test case of the TSP and fuzzy tuning for GLS. In addition, the value of the IPTS tuning method obtained can be further established by comparing its average performance relative to a simple approach whereby a fixed set of parameter values is being used.

The symmetric case of the optimisation problem considered calls for solving two-dimensional symmetric instances of the Traveling Salesman Problem (STSP). The STSP searches for the shortest Hamiltonian tour in a complete undirected graph $G = (V, E)$, visiting all vertices in V once, and given the length of each edge in E. Let $|V| = n$. For the STSP, the distance between two vertices $d(v_i, v_j) = d(v_j, v_i)$. The asymmetric Travelling Salesman Problem (ATSP) follows a similar definition by searching for the shortest tour in a complete directed Graph $G_D = (V, A)$, while $d(v_i, v_j) \neq d(v_j, v_i)$. The fuzzy tuning system is to work in front of the Guided Local Search (GLS) algorithm developed in Ries *et al.* (2012).

2.1 TSP instance characteristics

The instance-specific characteristics are the following four presented in Ries *et al.* (2012): the instance size n, the distance metric s, a clustering index c, and a shape ratio r. The number of vertices n is assumed given. The Minkowski distance metric determines s ($0 < s \leq \infty$), and an approximate value is typically available from the pratical context. In particular, $s = 1$ gives Manhattan and $s = 2$ gives Euclidean distances. These two cases correspond to major applications of the TSP (Applegate *et al.*, 2006, Schmitting, 1999). If s is not given, it takes the default value 1.5. The cophenetic correlation coefficient c ($0 \leq c \leq 1$) (Everitt *et al.*, 2001) is used to indicate the degree of clustering; the closer c is to 1, the more certain the instance is clustered. Finally, the shape of the area in which the vertices of an instance are distributed determines the value for r ($1 \leq r \leq \infty$). It is calculated as the ratio of the largest side to the smallest side of the rectangle with minimal area that encompasses all vertices. Finally, it is assumed that some appropriate value of p is provided by the decision maker.

2.2 Case study GLS for the TSP

GLS for TSP (Voudouris and Tsang, 1999) is based on the principle of iteratively calling a local search procedure with incrementally adapted distance data. In each iteration of GLS, the longest edge in the current solution is penalised by increasing its corresponding distance value. However, the penalisation principle also diversifies by taking into account the relative number of times an edge has been penalised. The particular GLS algorithm selected in this study incorporates 2-opt local search, and, to reduce computational effort, the mechanisms of active marking (Bentley 1992) and neighbour lists. The parameters of this algorithm are denoted by α ($0 < \alpha \leq 1$), NL ($0 < NL \leq 1$), and IT; α is the fraction of the average edge length in an initial solution with which an edge is penalised; NL determines the length of the sorted nearest neighbour list of every vertex, i.e. $(n-1)NL$ is the number of vertices in each list; and IT is the number of GLS-iterations.

The GLS algorithm is one of the better deterministic algorithms for solving symmetric TSPs (see Voudouris and Tsang 1999). It should be emphasised that the aim of IPTS is not to identify which metaheuristic is best, but to facilitate practioners in not having to set the values of parameters of a metaheuristic of which they may not have a detailed knowledge. The semi-automated approach aims to help designers of the IPTS to develop a tuning algorithm as to make the performance of a given metaheuristic as good as possible, while allowing the decision maker to express his preference concerning the time pressure.

2.3 Fuzzy IPTS

We shortly outline the fuzzy logic concepts used in the IPTS approach developed in Ries *et al.* (2012), and refer to this article for further details. Fuzzy logic, introduced by Zadeh (1965), is a set theory about objects for which partial set membership (any value between 0 and 1) rather than crisp set membership (either 0 or 1) is allowed. A Fuzzy Inference System (FIS) aims to specify an output from a given input by means of fuzzy logic. The input in this case are crisp values of instance characteristics n, s, c, and r, and the time preference p, and the different outputs are crisp values for each of the algorithm-specific parameters of the GLS. The FIS consists of membership functions, logical operations, and an *if-then* rule base. A membership function describes how a crisp value for one input (output) maps onto a membership value of a specific linguistic term associated with a valid set of the input (output). For example, the input n can be described by three sets labelled, respectively, 'small', 'medium', or 'large'; an instance of size $n = 500$ may hence be mapped to being 'small' of degree 0.4, and being 'medium' of degree 0.6. The process of translating crisp input to membership values is called fuzzification. This process depends on the shape of the set membership functions. The proposed fuzzy IPTS design uses triangular- and trapezoidal-shaped functions due to their simplicity in design. A similar process, called defuzzification, determines a crisp value for an output based on the knowledge of its set membership functions and values. To find set membership values of an output based on set membership functions

of inputs, a set of fuzzy rules, or rule base, is needed. A simple example of a rule base to find the set membership values for the output IT would be:

- IF n is small AND r is rectangular THEN IT= small
- IF n is small AND r is squared THEN IT= medium
- IF n is medium THEN IT= medium
- IF n is large AND c is small AND r is rectangular THEN IT= medium
- IF n is large AND c is small AND r is squared THEN IT= large
- IF n is large AND c is large THEN IT= large

The fuzzy AND operator will determine a combined degree of truth of the antecedent of each rule as the intersection (or minimum value) of all membership values of the inputs. In the presented design the centroid method is applied, returning the value corresponding to the centre of the area generated from all rules applying to that particular output.

This paper presents an approach for the automatic derivation of a rule base from an automatically generated decision tree using the TDIDT algorithm, further discussed in Section 3. The approach has, to the best of our knowledge, not yet been applied to the PSP problem, but has been applied in many other applications. For example, Sugumaran & Ramachandran (2007) use decision tree rule extraction to design a fuzzy system applied in fault diagnosis.

2.4 Experimental set-up

The experimental set-up follows that of Ries *et al.* (2012) and in fact, reuses the same training sets $\Phi_{GLS-STSP}$ and $\Phi_{GLS-ATSP}$ of instances. These instances were generated by a random instance generator such that each instance characteristic, see Table 1, forms an input parameter. It is noted that in the case of the ATSP the parameter distance metric has been excluded. For non-clustered instances a set of randomly distributed points were generated

within a rectangle specified by r. In the case of clustered instances, the approach of Johnson and McGeoch (2002) is adapted by choosing a set of centre locations and creating a set of data points that are normally distributed around a selected cluster centre. There are in total 6 instances in each of 192 classes. These classes were based on a full-factorial design for the factors n, s, r, c, α, NL, and IT, see Table 1. All factors were considered at two levels only, except for n for which three levels were used.

Table 1: 2-Factorial Design

	- 1	+ 1
Number of vertices* n	100	1000
Clustering c	Non-Clustered (0)	Clustered (1)
Distance metric** s	Manhattan Distance (1)	Euclidean Distance (2)
Ratio r	1 (Square)	100 (Rectangle)
GLS-Alpha α	0.2	0.4
GLS-NL-Size NL	0.2	0.4
GLS-Iterations IT	1000	100000

*Vertices on third level '0' with 500 vertices, **excluded for ATSP

For each of the instances in $\Phi_{GLS-STSP}$ and $\Phi_{GLS-ATSP}$, values need to be obtained for solution quality and computational time. As in Ries *et al.* (2012), the experiments on each TSP instance are conducted in a way that reduces the number of needed runs as follows. For each of the factor value combinations for α and NL for GLS as specified in Table 1, one run is conducted up to 3 million GLS iterations for the symmetric case and 50000 GLS-iterations for the asymmetric case. At the start of each of these runs (four runs in total), the length of the initial TSP tour is recorded, $f^0(GLS)$, which is the length obtained from the repeated nearest neighbour heuristic. When the number of GLS-iterations reaches one of the specified factor values for GLS-iterations listed in Table 1, the length of the incumbent TSP tour $f(GLS)$ and the computational time consumed is recorded. At the end of the run, the length of the final incumbent solution is recorded. The TSP tour of minimum length across all four runs is calculated and called $f_h^*(GLS)$.

The above approach is possible when having access to the metaheuristic code. For meta-heuristics only available as a black-box, each of the 1152 instances would need to be separately

ran, as well as (perhaps) an additional series of long runs to establish values for $f_h^*(.)$.

The measure of the solution quality of a given TSP instance when using algorithm a and algorithm parameter combination $P(a)$ as proposed in Ries *et al.* (2012) is adopted, and defined as:

$$S(a, P(a)) = (f(a, P(a)) - f_h^*(a))/(f^0(a) - f_h^*(a)). \quad (1)$$

This measure hence scales the solution quality S of a given instance in the training sets $\Phi_{GLS-STSP}$ and $\Phi_{GLS-ATSP}$ to a value between 0 and 1. The lower S, the better the particular algorithm-specific parameter combination is for solving this TSP instance.

Table 2: Membership Subsets - Input Variables

Parameter	Subset	Min	Med1	Med2	Max
Number of vertices n	Small	0	0	400	800
Number of vertices n	Medium	400	800	-	1200
Number of vertices n	Large	800	1200	5000	5000
Clustering c	Non-Cl	0	0	-	1
Clustering c	Cl	0	1	-	1
Distance metric s	Man	0	0	-	2
Distance metric s	Eucl	1	2	-	3
Ratio r	Square	0	0	10	100
Ratio r	Rect	10	100	150	150
Time pressure p	Small	0	0	-	0.5
Time pressure p	Medium	0	0.5	-	1
Time pressure p	Large	0.5	1	-	1

The design of input and output set membership functions for a fuzzy system is an intuitive approach which in the presented case is based on the factorial design. Tables 2 and 3 show the membership functions adopted in this study for the STSP. While the input parameter membership functions are kept the same for the ATSP, Table 4 shows the structure of the membership functions for the ATSP. It is a well-known issue that the design of the membership functions affects the performance of the fuzzy inference system. However, no further analysis is conducted in this paper related to this matter.

Table 3: Membership Subsets - Output Variables - GLS - STSP

Parameter	Subset	Min	Med I	Med II	Max
GLS-Alpha α	Small	0	0.2	-	0.4
GLS-Alpha α	Large	0.2	0.4	-	0.6
GLS-NL-Size NL	Small	0	0.2	-	0.4
GLS-NL-Size NL	Large	0.2	0.4	-	0.6
GLS-Iterations IT	Small	0	0	10000	50000
GLS-Iterations IT	Large	25000	100000	240000	240000

Table 4: Membership Subsets - Output Variables - GLS - ATSP

Parameter	Subset	Min	Med I	Med II	Max
GLS-Alpha α	Small	0	0.2	-	0.4
GLS-Alpha α	Large	0.2	0.4	-	0.6
GLS-NL-Size NL	Small	0	0.2	-	0.4
GLS-NL-Size NL	Large	0.2	0.4	-	0.6
GLS-Iterations IT	Small	0	0	10000	50000
GLS-Iterations IT	Large	10000	50000	240000	240000

3 Rule extraction using TDIDT

3.1 Decision tree classification

Besides neural networks and nearest neighbour classifiers, decision tree modelling is one of the most popular approaches in the area of classification. The aim of (crisp) classification is to assign each object of a series to exactly one of several classes according to object features or attributes. Formally, a classification problem is defined as the mapping $g : O \rightarrow C$ of a set of objects $O = \{o_1, o_2, ..., o_n\}$, each of which is characterised by a set of attributes, on a set of classes $C = \{c_1, c_2, ..., o_k\}$ such that each object is assigned to exactly one class.

A decision tree used for classification is a tree where the root node and each internal node is labelled with a question related to (typically one of) the attributes, see also Figure 2 for a simple example. Each arc eminating from each such node represents one of a finite set of possible answers to the associated question, with the set of arcs eminating from a node comprising the complete set of possible answers. A node that does not have any arcs eminating from it is called a leaf node and refers to a particular class, with the set of all

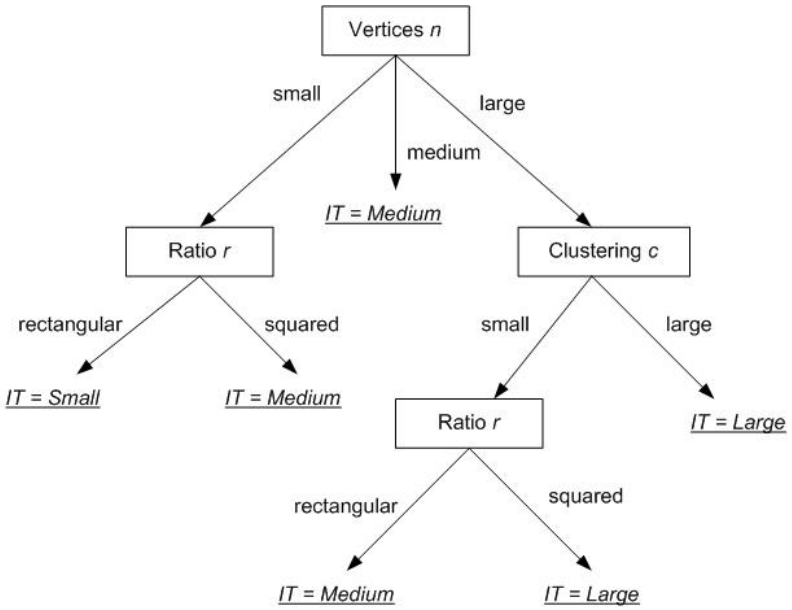

Figure 2: Decision Tree Example

leaf nodes referring to all classes of set C at least once. Given a decision tree, each object of the set O can be examined one by one, and by answering the questions subsequently found at each node and following the corresponding branch in the tree, its classification obtained. From a decision tree, a set of corresponding classification rules can be derived. For example, the rule base presented in Section 2.3 corresponds to the decision tree of Figure 2.

3.2 Decision tree design using TDIDT

Decision tree classification is mainly a 2-stage approach (Dunham, 2003, p.73). Firstly, a training set of objects is used to design a decision tree structure and, secondly, a test set of objects is classified according to the designed structure to assess its classification power. To design the tree, this paper applies the Top-Down Induction of Decision Tree (TDIDT) algorithm (see Quinlan, 1979), using in particular the inducer rule induction workbench by Bramer (2004) with as attribute selection method the information gain concept.

Given a training set of objects, a list of considered categorical attributes about each of these objects, and knowledge about the class to which each object belongs, the TDIDT

algorithm employs a top-down greedy search through the space of possible decision trees to find the best possible tree satisfying the property that all objects in any of its leaf nodes belong to the same class. The most important part of the algorithm is to decide on how to create child nodes from every node in the tree. This is often translated into deciding on which attribute to split at each node, and to create the corresponding branches according to the attribute values. This procedure is also called recursive partitioning (Bramer, 2007).

Popular attribute selection criteria include the information gain (Quinlan, 1979) and the gini index (Breiman *et al.*, 1984). The former is based on the concept of entropy E, a measure of uncertainty from information theory (Shannon, 1948). The entropy of a node L_i in a tree is:

$$E(L_i) = \sum_{j=1}^{k} -p(j|L_i)\log_2 p(j|L_i) \tag{2}$$

where $p(j|L_i)$ represents the relative frequency of objects in the training set of class c_j at node L_i (only considering classes with $p(j|L_i) \neq 0$), and is calculated by dividing the number of objects in c_j at node L_i by the total number of objects at node L_i. If from among the set of attributes, attribute A with categorical values $\{A_1, ..., A_v\}$ is selected to split upon and derive v child nodes $\{L_{i+1}, ..., L_{i+v}\}$, the information gain $G(A)$ from splitting the parent node L_i into v partitions through A is the expected reduction of the entropy:

$$G(A) = E(L_i) - \sum_{m=1}^{v} \frac{l_{i+m}}{l_i} E(L_{i+m}) \tag{3}$$

where l_i and l_{i+m} is the number of objects in L_i and L_{i+m}, respectively ($m = 1, ..., v$), and the second term is the expected entropy presented in the collection of child nodes $\{L_{i+1}, ..., L_{i+v}\}$. According to the information gain criterion, the best attribute A to split upon would be that one which will maximise (3).

3.3 Input requirements and validation approach

The TDIDT algorithm requires all included attributes to be categorical data instead of continuous. Standard discretisation approaches partition the range of continuous data into a number of categories. Often used approaches are the equal-width-interval method and the equal-frequency-interval method. In the first method, the continuous data is subdivided into t categories by division of the total range by t. The second method places the boundaries between the categories as to obtain an equal number of values in each category.

There are several possible methods to validate the derived decision tree and the corresponding classification rules. This study uses a cross validation (Han & Kamber, 2006), a popular approach that separates the data set of n instances into k different equal sized subsets. (k-1) of the subsets are used to design a decision tree and one is used as test set to evaluate the decision tree. This process is repeated k times such that each subset is at least once used as a test set. The estimated accuracy is then calculated by the number of correct classification overall k runs and is divided by the total number of data instances.

3.4 Application to the test case

In decision tree terminology, each instance in Φ_{GLS} is an object, and each of the factors within the factorial design outlined by Table 1 an object attribute. Given that the design of the instance set follows the factorial set-up, the attributes are categorical in nature.

For preferences *p is small* and *p is large*, solution quality (S) and computational time (T) are, respectively, determining the classification of the objects. As these are recorded as continuous data they need to be discretised. The case study uses a subjective approach to introduce a set of classes representing different levels of S, T and Balance (B) - a combination of both. Table 5 shows the discretisation intervals for Solution Quality and Computational Time.

A decision maker preference describing a balance of both performance measures is derived

Table 5: Discretisation - Heuristic Performance

S (Excess in %)	0 to < 0.01	0.01 to < 1	1 to < 5	5 to < 15	15 to < 50	≥ 50
Class - S	1	2	3	4	5	6
T (in seconds)	0 to < 1	1 to < 3	3 to < 10	10 to < 25	25 to < 50	≥ 50
Class - Time	1	2	3	4	5	6

by taking an average of both assigned class intervals. Therefore, the number of classes for the discretisation of Balance ranges from 1, 1.5, 2, 2.5,..6. For example, the combination of a good solution quality (Class 2 - S) associated with a reasonably long computational time (Class 4 - T) would result in a B Class 3.

Once all results for the set Φ_{GLS} are discretised according to S, T and B, the inducer rule induction workbench by Bramer (2004) is used to derive a set of rules for the algorithm-specific parameters, α, NL and IT, using the set of instance-specific parameters and a performance class representing a decision maker preference. Hence, a set of decision rules is derived for each decision maker preference in combination with each algorithm-specific parameter. For the set of rules corresponding to best solution qualities, all rules including Class 1 - SQ are selected. Similarly, rules including Class 1 - T are selected to represent decision maker preference on short computational times. The balance is explained by the set of rules including Class 2.5 - B.

It is important to note that based on the experimental set-up and, for example, the fixed upper level of IT to 100000, does not allow for every single combination of instance characteristics to achieve Class 1 - S and, subsequently, Class 2.5 - B. Hence, for some sets of instance characteristics, rules were adjusted by using those rules that are associated with a performance level closest to the anticipated one: Best S (p=0), Balance (p=0.5), Short T (p=1), as outlined in Table 6 and Table 7 for the STSP and ATSP, respectively.

Table 6 shows the automatically derived set of rules for each decision maker preference and each algorithm-specific parameters. The predictive accuracy for the STSP is given in Table 8 for each combination of algorithm-specific parameter and investigated decision maker preference. It shows that the obtained classification rules for α are of reasonable predictive

Table 6: Rule-Base obtained by TDIDT - GLS - STSP

Input				p=0			p=0.5			p=1		
n	c	s	R	α	NL	IT	α	NL	IT	α	NL	IT
Small	Non-Clust	Man	Square	s	l	l	l	s	s	s	s	s
Small	Non-Clust	Man	Rect	s	l	l	l	s	s	s	s	s
Small	Non-Clust	Eucl	Square	l	s	l	s	s	s	s	s	s
Small	Non-Clust	Eucl	Rect	s	s	l	s	s	l	s	s	s
Small	Clust	Man	Square	s	l	l	l	l	s	s	s	s
Small	Clust	Man	Rect	s	l	l	s	l	s	s	s	s
Small	Clust	Eucl	Square	l	l	l	s	s	s	s	s	s
Small	Clust	Eucl	Rect	s	l	l	s	l	s	s	s	s
Medium	Non-Clust	Man	Square	s	s	l	l	s	s	s	s	s
Medium	Non-Clust	Man	Rect	s	s	l	s**	s**	l**	s	s	s
Medium	Non-Clust	Eucl	Square	s	s	l	l	s	s	s	s	s
Medium	Non-Clust	Eucl	Rect	s	s	l	s**	s**	s	s	s	s
Medium	Clust	Man	Square	s	s	l	s	s	s	s	s	s
Medium	Clust	Man	Rect	s	s	l	s	l	s	s	s	s
Medium	Clust	Eucl	Square	l	s	l	s	s	s	s	s	s
Medium	Clust	Eucl	Rect	s	s	l	l	l	s	s	s	s
Large	Non-Clust	Man	Square	s*	s*	l	l	l	s	s	s	s
Large	Non-Clust	Man	Rect	s	s	l	s	l	s	s	s	s
Large	Non-Clust	Eucl	Square	l	s	l	l	l	s	s	s	s
Large	Non-Clust	Eucl	Rect	s	s	l	l	l	s	s	s	s
Large	Clust	Man	Square	s	s	s	l	s	s	s	s	s
Large	Clust	Man	Rect	s	l*	l*	s	l	s	s	s	s
Large	Clust	Eucl	Square	l	l	l	l	s	s	s	s	s
Large	Clust	Eucl	Rect	s	l	l	l	l	s	s	s	s

*uses Class 3 - SQ, **uses Class 3 - Balance

Table 7: Rule-Base obtained by TDIDT - GLS - ATSP

Input			p=0			p=0.5			p=1		
n	c	R	α	NL	IT	α	NL	IT	α	NL	IT
Small	Non-Clust	Square	s	l	l	s	l	s	s	l	s
Small	Non-Clust	Rect	s	l	l	s	l	s	s	l	s
Small	Clust	Square	s	l	s	s	l	l	s	l	s
Small	Clust	Rect	l	l	l	s	l^{***}	l	s	l	s
Medium	Non-Clust	Square	s	l	l	s	l^{***}	s	l	s	s
Medium	Non-Clust	Rect	s	l	l	l	l^{***}	s	l	s	s
Medium	Clust	Square	s	l	l	s	l	l	s	l	s
Medium	Clust	Rect	s	l	l	s	l^{***}	l	s	l	s
Large	Non-Clust	Square	s	l	l^{*}	l^{**}	l	s	s	s	s
Large	Non-Clust	Rect	s	l	l^{*}	s	s^{***}	s	s	s	s
Large	Clust	Square	s	l	l^{*}	s	l	s	s	s	s
Large	Clust	Rect	s	l	l^{*}	s^{**}	l^{***}	s	s	s	s

*uses Class 2 - SQ, **uses Class 3 - Balance, ***uses Class 2 - Balance

power, while the level of prediction accuracy is higher for parameters NL and IT. This is similarly shown for the ATSP in Table 9.

Table 8: Predictive accuracy GLS - STSP

Algorithm-specific parameter	p=0	p=0.5	p=1
GLS-Alpha α	0.61	0.62	0.53
GLS-NL-size NL	0.67	0.7	0.63
GLS-Iterations IT	0.8	0.88	1.0

Table 9: Predictive accuracy GLS - ATSP

Algorithm-specific parameter	p=0	p=0.5	p=1
GLS-Alpha α	0.53	0.6	0.54
GLS-NL-size NL	0.68	0.73	0.77
GLS-Iterations IT	0.78	0.85	0.97

4 Performance comparison

Computational experiments have been run for a set Φ_{STSP} of 50 TSPLIB instances (Reinelt, 1991) and a randomly created set of 35 ATSP instance Φ_{ATSP} and three different decision maker preferences: p=0, p=0.5 and p=1.

The solution quality SQ for instance ϕ_i from the test sets Φ_{STSP} and Φ_{ATSP} is calculated as the percentage of tour length obtained by the implemented fuzzy logic IPTS $f(\pi^{Fuzzy}, \phi_i)$ excessing the known optimal tour length $f^{Opt}(\phi_i)$:

$$SQ(\pi^{Fuzzy}, \phi_i) = (f(\pi^{Fuzzy}, \phi_i) - f^{Opt}(\phi_i))/(f^{Opt}(\phi_i))$$

Computational times T are reported in absolute terms (seconds).

Table 10 shows the average performance of Φ_{STSP} for each decision maker preference derived by the Fuzzy IPTS, compared to a range of fixed parameter settings derived from all combinations of α = 0.2, 0.3, 0.4, NL=0.2, 0.3, 0.4 and IT = 1000, 10000, 50000, 100000, 200000. It is important to recognise that within the given factorial design in Table 1 the

level set for IT does not exceed 100000, while the the middle level of 0.3 is not considered for α and NL.

Considering the reduction of computational time, results show on average an excess of 0.67% compared to the optimal solution known while computational time is 2.07 seconds. With regards to decision maker preference on short computational times and a balance between SQ and Time, the average computational time values for $p = 0$ and $p = 0.5$ differ marginally, similar to the average solution quality.

Table 10: GLS-Performance - Φ_{STSP}

IPTS using decision trees						
Decision Maker preference			SQ - Excess Optimality	SQ-Stdev	Time	Time-Stdev
SQ			0.0027	0.0052	19.865	22.5280
Balance			0.0064	0.0093	2.3478	2.7256
Time			0.0067	0.0096	2.0732	2.2653
IPTS using manual set-up (Ries *et al.*, 2012)						
SQ			0.0018	0.0038	48.56	
Balance			0.0028	0.0064	22.26	
Time			0.0083	0.0108	2.27	
Fixed parameter setting						
NL	Alpha	IT	SQ - Excess Optimality	SQ-Stdev	Time	Time-Stdev
0.2	0.2	1000	0.0263	0.0242	0.1737	0.1809
0.2	0.3	1000	0.0231	0.0237	0.1797	0.1917
0.2	0.4	1000	0.0219	0.0223	0.1857	0.1990
0.2	0.2	10000	0.0096	0.0128	1.1548	1.2650
0.2	0.3	10000	0.0084	0.0114	1.1822	1.2972
0.2	0.4	10000	0.0082	0.0102	1.2153	1.3385
0.2	0.2	50000	0.0042	0.0069	5.4509	6.0464
0.2	0.3	50000	0.0034	0.0059	5.5856	6.2406
0.2	0.4	50000	0.0043	0.0072	5.7167	6.4171
0.2	0.2	100000	0.0029	0.0052	10.9188	12.2247
0.2	0.3	100000	0.0024	0.0052	11.1882	12.5729
0.2	0.4	100000	0.0029	0.0058	11.3109	12.7544
0.2	0.2	200000	0.0017	0.0040	22.0182	24.9092
0.2	0.3	200000	0.0014	0.0030	22.3496	25.4003
0.2	0.4	200000	0.0022	0.0048	22.5647	25.6598
0.3	0.2	1000	0.0254	0.0242	0.2021	0.2188
0.3	0.3	1000	0.0229	0.0230	0.2109	0.2351
0.3	0.4	1000	0.221	0.0213	0.2154	0.2394
0.3	0.2	10000	0.0095	0.0119	1.3647	1.4976
0.3	0.3	10000	0.0081	0.0109	1.435	1.5817
0.3	0.4	10000	0.0087	0.0107	1.4737	1.6278
0.3	0.2	50000	0.0049	0.0077	6.66	7.4027
0.3	0.3	50000	0.0038	0.0063	6.9149	7.7149
0.3	0.4	50000	0.0037	0.0061	7.0527	7.8963
0.3	0.2	100000	0.0035	0.0065	13.4044	15.0062
0.3	0.3	100000	0.0026	0.0047	13.8559	15.5834
0.3	0.4	100000	0.0028	0.0053	14.0043	15.8256
0.3	0.2	200000	0.0024	0.0058	27.1891	30.6628
0.3	0.3	200000	0.0019	0.0039	27.9291	31.5907
0.3	0.4	200000	0.0024	0.0052	28.1212	31.9135
0.4	0.2	1000	0.0251	0.0234	0.2347	0.2602
0.4	0.3	1000	0.022	0.0230	0.2537	0.2813
0.4	0.4	1000	0.0217	0.0213	0.2522	0.2881
0.4	0.2	10000	0.0092	0.0120	1.6297	1.7933
0.4	0.3	10000	0.0077	0.0106	1.7384	1.9176
0.4	0.4	10000	0.0086	0.0104	1.7875	1.9655
0.4	0.2	50000	0.004	0.0064	8.0614	8.9159
0.4	0.3	50000	0.0035	0.0059	8.3398	9.2742
0.4	0.4	50000	0.0038	0.0062	8.3612	9.3316
0.4	0.2	100000	0.0027	0.0053	16.0343	17.8566
0.4	0.3	100000	0.0021	0.0039	16.4925	18.5646
0.4	0.4	100000	0.0029	0.0054	16.7156	18.8145
0.4	0.2	200000	0.0018	0.0046	32.2582	36.2234
0.4	0.3	200000	0.0017	0.0037	33.0849	37.4601
0.4	0.4	200000	0.0025	0.0052	33.5938	38.0602

In comparison to the selected set of fixed parameter settings (diamonds), it can be seen

in Figure 3 that the results associated with the fuzzy IPTS can be found together with other fixed parameter combinations on the Pareto front or reasonably close in the case of the preference on good solution quality. In contrast to the IPTS approach (triangular shaped), all fixed parameter settings are chosen randomly.

Figure 3 shows that in particular the right-hand side ending of the curve stagnates: the best level of solution quality is reached with a computational time of 20 seconds and is not significantly improved for any larger computational times. On the other end, a solution quality of 1% can be considered as a large value for solution quality that is still considered to be a reasonable performance; any value above may be disregarded as not acceptable. The semi-automated IPTS results can be found within the non-stagnant parts or good parts of the curve. Solution quality can be kept below 0.7% while the manual set-up moves from either above that value for very short T to below 0.5% but above 5 seconds. Hence, the semi-automated IPTS has shown potential to interpolate heuristic performance along the Pareto curve.

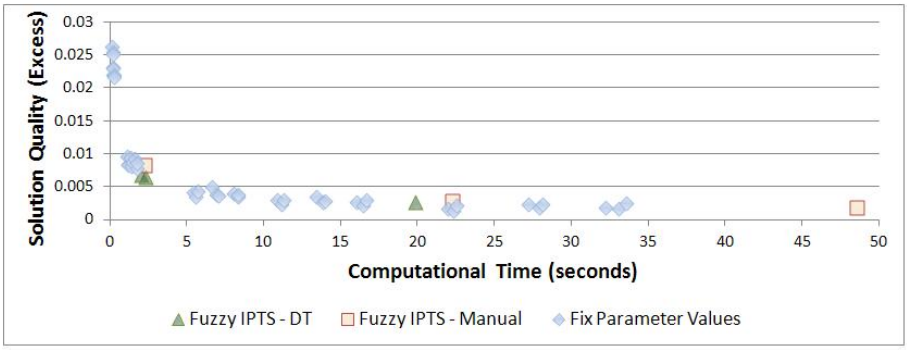

Figure 3: IPTS settings and Fixed settings - STSP

Figure 3 also indicates that parameter settings obtained by a manual set-up of a fuzzy IPTS (circular shaped) using statistical insights of effects between instance characteristics and algorithm-specific parameters result in a reasonable performance compared to a set of randomly chosen fixed set of parameter values. A manual approach requires, however, a more detailed understanding and expertise with the particular algorithmic behaviour which

not every decision maker may be equipped with. Hence, non-experts in meta-heuristics benefit from an automated approach that replaces a substantial statistical analysis needed in a manual approach (Ries *et al.*, 2012).

Table 11 shows the average performance of Φ_{ATSP} for each decision maker preference derived by the Fuzzy IPTS. A comparison is made to a set of fixed parameter settings namely $\alpha = 0.2, 0.4$, NL=0.2, 0.4 and $IT = 1000, 10000$. This is due to observations made in preliminary testing that have shown the consistent behaviour of limited improvement beyond 50000 iterations.

Table 11: GLS-Performance - Φ_{ATSP}

IPTS using decision trees						
Decision Maker preference			SQ - Excess Optimality	SQ-Stdev	Time	Time-Stdev
SQ			0.1678	0.2571	123.02	115.89
Balance			0.1165	0.3186	95.34	57.76
Time			0.1727	0.3358	13.34	8.71
Fixed parameter setting						
NL	Alpha	IT	SQ - Excess Optimality	SQ-Stdev	Time	Time-Stdev
0.2	0.2	1000	0.2516	0.2528	7.6592	8.0063
0.2	0.2	10000	0.1210	0.1932	62.1746	55.4399
0.4	0.2	1000	0.3799	0.3422	6.2861	6.0012
0.4	0.2	10000	0.2497	0.3209	47.9674	39.1947
0.2	0.4	1000	0.3127	0.3174	12.7901	12.1438
0.2	0.4	10000	0.2167	0.2669	97.0381	80.8173
0.4	0.4	1000	0.2905	0.3224	12.9115	12.3110
0.4	0.4	10000	0.1918	0.2896	102.7516	86.3560
0.3	0.3	1000	0.3059	0.3335	9.8195	9.3275
0.3	0.3	10000	0.2286	0.2720	76.8627	65.7613
0.2	0.3	1000	0.3308	0.3393	9.7246	9.3087
0.2	0.3	10000	0.2295	0.2689	75.7255	65.9128
0.4	0.3	1000	0.2985	0.3293	9.8553	9.3421
0.4	0.3	10000	0.2268	0.2920	78.5166	67.1043
0.3	0.4	1000	0.2921	0.3214	12.7923	12.3339
0.3	0.4	10000	0.1844	0.2594	103.4103	87.7988
0.3	0.2	1000	0.3796	0.3407	6.3314	6.0966
0.3	0.2	10000	0.2719	0.3135	49.7673	42.1308

Figure 4 shows that the SQ levels for all decision maker preferences are fairly similar. It also underlines the importance of the impact of membership functions as an anomaly can be observed with the p = 0.5 showing a better SQ than p = 0. It shows the sensitivity of the approach to the extraction of the rule base from the decision tree. In Figure 4 several decisions had been made to determine a corresponding rule if the performance class (Balance = 2.5) is not existing for a combination of instance-specific characteristics. This manual consideration impact final performance and in the presented case has been made with a preference to SQ.

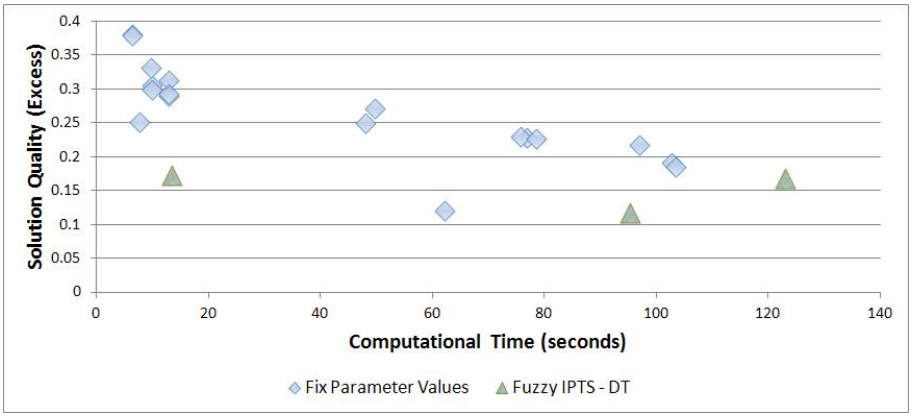

Figure 4: IPTS settings and Fixed settings - ATSP

5 Conclusions

The use of classification rule extraction using decision trees to automatically design a fuzzy IPTS rule base has been introduced. This approach is semi-automated due to the facilitation of the rule base design in addition to the manual design of membership subsets in a fuzzy system. The success of the presented method is considerably influenced by the discretisation of both objective variables - solution quality and computational time, including the adjusted combined performance value that represents a decision maker preference in heuristic performance as a combination of good solution quality and short computational times.

The performance of the TDIDT design is also dependent on the mode of attribute selection. The most influential factor, however, is likely to be the investigated data set Φ_a for an algorithm a itself. As it is based on a full factorial design, it has resulted in a small number of membership subsets. The level of flexibility may be increased by extending the data set to a larger number of levels in the factorial design.

The results are promising and provide a structured technique of creating a rule base with the flexibility of modifying the corresponding membership functions. A semi-automated IPTS using fuzzy logic shows potential in finding a set of parameter values that results in good heuristic performance, according to a decision maker preference. The presented system is static in its design such that once the set of rules and set of membership functions are

constructed they are fixed. An adaptive system to differently structured instances using an evolving design strategy is currently being investigated.

Appendix

The test set Φ_{TSP} consists of the following TSPLIB instances: a280,berlin52, bier127, ch130, ch150, d1291, d1655, d198, d493, d657, dsj1000, eil101, eil51, eil76, fl1400, fl1577, fl417, kroA100, kroB100, kroB150, kroB200, kroC100, kroD100, kroE100, lin105, linhp318, nrw1379, p654, pcb1173, pr107, pr124, pr136, pr144, pr152, pr226, pr264, pr299, pr439, pr76, rat195, rat99, rd100, rd400, rl1304, rl1323, rl1889, ts225, u1060, u1432, u159, u1817.

References

[1] B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

[2] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook. *The Travelling Salesman Problem - A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, 2006.

[3] R.S. Barr, B.L. Golden, J. Kelly, W.R. Stewart, and M.G.C Resende. Guidelines for designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[4] R. Battiti. Reactive search: Toward self-tuning heuristics. In I.H. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern heuristic search methods*, pages 61–83. John Wiley and Sons Ltd., New Jersey, 1996.

[5] J.J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[6] M. Bramer. Inducer: a rule induction workbench for data mining. In *In Proceedings of the 16th IFIP World Computer Congress Conference on Intelligent Information Processing*, pages 499–506, Beijing, 2004.

[7] M. Bramer. *Principles of data mining.* Undergraduate topics in computer science. Springer, London, 2007.

[8] S. P. Coy, B. L. Golden, G. C. Runger, and E. A.‘ Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

[9] M.H. Dunham. *Data Mining: Introductory and Advanced Topics.* Prentice Hall, New Jersey, 2003.

[10] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[11] B. S. Everitt, S. Landau, and M Leese. *Cluster Analysis.* Arnold, London, 4th edition, 2001.

[12] C.A. Gil, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. *Constraint Programming 2009, Springer LNCS 5732*, pages 142–157, 2009.

[13] J. Han and M. Kamber. *Data mining - Concepts and Techniques.* Morgan Kaumann Publishers, San Francisco, 2nd edition, 2006.

[14] J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

[15] F. Hutter, H.H. Hoos, and T. Stuetzle. Automatic algorithm configuration based on local search. In *Twenty-Second Conference on Artifical Intelligence (AAAI ’07)*, pages 1152–1157, Menlo Park, 2007. AAAI Press.

[16] S.-J. Jeong, K.-S. Kim, and Y.-H. Lee. The efficient search method of simulated annealing using fuzzy logic controller. *Expert Systems with Applications*, 36(3):5, 2009.

[17] D.S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M.H. Goldwasser, D.S. Johnson, and C.C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: In Proceedings of the 5th and 6th DIMACS Implementation Challenges*, pages 215–250, Providence, 2002. American Mathematical Society.

[18] D.S. Johnson and L.A. McGeoch. Experimental analysis of heuristics for the stsp. In G. Gutin and A.P. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer: Academic Publishers, Norwell, 2002.

[19] S. Kadioglu, Y. Malitsky, M. Sellmann, , and K. Tierney. An instance-specific algorithm configuration. In *In Proceedings of the 19th European Conference on Artificial Intelligence*, pages 751–756, Amsterdam, 2010. IOS Press.

[20] M. Kern. *Parameter Adaption in Heuristic Search - A Population-Based Approach -*. Phd thesis, University of Essex, 2006.

[21] R. Pavón, F. Díaz, R. Laza, and V. Luzón. Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *Expert Systems with Applications*, 36(2):3407–3420, 2009.

[22] J.R. Quinlan. Discovering rules form large collections of examples: A case study. In D. Michie, editor, *Expert systems in the micro electronic age*. Edinburgh Press, Edingburgh, 1979.

[23] G. Reinelt. Tsplib - a traveling salesman problem library. *Journal of Computing*, 3(4):376–384, 1991.

[24] E. Ridge. *Design of Experiments for the Tuning of Optimisation Algorithms*. Phd thesis, University of York, 2007.

[25] J. Ries, P. Beullens, and D. Salt. Instance-specific multi-objective parameter tuning based on fuzzy logic. *European Journal of Operational Research*, 218(2):305–315, 2012.

[26] W. Schmitting. *Das Traveling Salesman Problem: Anwendung und heuristische Nutzung von Voronoi-/Delaunay-Strukturen zur Loesung euklidischer, zweidimensionaler Traveling-Salesman-Probleme*. PhD thesis, University of Duesseldorf, 1999.

[27] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 623–656, 1948.

[28] V. Sugumaran and K.I. Ramachandran. Automatic rule learning using decision tree for fuzzy classifier in fault diagnosis of roller bearing. *Mechanical Systems and Signal Processing*, 21(5):2237–2247, 2007.

[29] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, 1999.

[30] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.