Learning Classifier Systems for Optimisation Problems: A Case Study on Fractal Travelling Salesman Problem

Maximiliano Tabacman
Departamento de Computacion, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina
tabacman@dc.uba.ar

Natalio Krasnogor[*]
ASAP research group, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
natalio.krasnogor@nottingham.ac.uk

Jaume Bacardit
ASAP research group, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
Multidisciplinary Centre for Integrative Biology, School of Biosciences, University of Nottingham, Sutton Bonington, LE12 5RD, UK
jaume.bacardit @nottingham.ac.uk

Irene Loiseau
Departamento de Computacion, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina
irene@dc.uba.ar

ABSTRACT

This paper presents a set of experiments on the use of Learning Classifier Systems for the purpose of solving combinatorial optimisation problems. We demonstrate our approach with a set of Fractal Travelling Salesman Problem (TSP) instances for which it is possible to know *by construction* the optimal tour regardless of the number of cities in them. This special type of TSP instances are ideal for testing new methods as they are well characterised in their solvability and easy to use for scalability studies. Our results show that an LCS is capable of learning rules to recognise to which family of instances a set containing a sample of the cities in the problem belongs to and hence automatically select the best heuristic to solve it. Moreover, we have also trained the LCS to recognise links belonging to the optimal tour.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning, Induction*; G.1.6 [**Numerical Analysis**]: Optimization

General Terms

Algorithms, Experimentation, Performance

Keywords

traveling-salesman-problem, optimization, learning-classifier-systems, machine learning

[*]Corresponding author

1 Introduction

In [8, 5] Moscato and Norman proposed a family of TSP instances for which one can know by construction the optimal tour for an arbitrary large number of cities. This property makes these TSP instances very appealing for testing new heuristics, approximate and exact methods and also scalability properties. Moreover, in [7] the authors also demonstrated theoretically that some of these instances can be solved optimally by a heuristic method. For example they showed that the so called MPeano (displayed in figure 2) fractal TSP can always be solved by both *Nearest Neighbour* and *Multiple Fragment* heuristics, whereas MNPeano (figure 2) can only be optimally solved by *Multiple Fragment*. Both of these algorithms might fail when trying to solve David Tour (figure 1) and Koch Tour (figure 2). On the other hand, they showed that *Furthest Addition From Minimal Convex Hull* can solve Koch Tour.

In this paper we investigate the possibility of using a Learning Classifier System [4, 10] to learn to classify (partial) instances as belonging to one of these fractal TSP families. The idea behind our research is, first, that if one could learn to perform this classification reliably then it would be possible to automatically choose which heuristic (from the set investigated by Moscato and Norman) to use for the given instance. We analyse these issues taking into consideration several descriptors for the sample instances and we also evaluate the impact of noise in our classification. Secondly, as a further proof-of-concept that the rules learnt by the LCS could be used with an optimisation goal in mind, we also train the LCS to recognise, for a set of TSP instances, which edges belong to the optimal tour and which do not belong to it. We finally integrate these predictive models into a simple heuristic to solve the TSP problem. Although the results presented here can only be considered preliminary in nature they are very encouraging and could potentially lead to a new way of linking data mining and optimisation strategies.

2 Theoretical Background

This work focuses on a family of TSP instances known as **Fractal TSP** (FTSP) which were defined and characterised in [7, 5, 8, 6].

Instances of the TSP are built by iterating over the rewriting rules (axioms) of an L-System. A TSP instance of order n is obtained by iterating the L-system n times. Higher orders of n give rise to larger and more intricate city distributions. FTSP instances have the following properties:

1. By construction, for each instance of order n its optimal tour is known. This makes these instances ideal for studies related to the scalability of TSP heuristics, approximation and exact algorithms.

2. It has been proved (and elegantly summarized in Table 1 of [7]) that some heuristics can solve to optimality a subset of the FTSP considered in this work. Also, the authors demonstrate that none of the analysed heuristics can solve all the fractal families and symmetrically, not every family is solved by all heuristics:

   "We have seen that while some constructive heuristics reliably solve one such instance they may fail on another instance. Thus, we can view heuristics as set of indices which characterise an instance and our instances as indices which characterise heuristics".

As an example, consider the construction of a David Tour for the first six orders: the FTSP referred to as David Tour is a combination of Sierpinski arrowheads that starts with a shape based on an equilateral triangle. The iterative process used to obtain each new order implies adding new triangles and rotating them as specified in [7, 9]. The interpretation of F, X, and other symbols is the standard L-System nomenclature, that is F draws a line in the direction the "turtle" is facing, + increases the angle by $\frac{1}{6}$ of a turn (360/6 degrees), - decreases the angle by the same amount, X is part of an axiom without direct graphical meaning, and ! indicates that the angle considered is to be negated.

Following is a detail of the L-System needed to build David Tour instances as a function of n (the order):

```
Begin: FX-XFX-XFX-XFX-XFX-XF
Replacement Rules:
F = !F!-F-!F!
X = !X
```

In figure 1 six David Tour instances are shown for the first six orders. For example, at order 0 the instance is composed of the six cities displayed in the top left panel, while the order 1 David Tour contains 18 cities for which the optimal tour is depicted. The order 5 of Koch, MPeano and MNpeano tours is shown in figure 2.

3 Classifying TSP Instances

We use Learning Classifier Systems to learn to identify FTSP classes. We have tested two different approaches to tackle this problem. In the first one, labelled *Global Statistics*, we characterize instances by a global metric, and try to learn how to predict the class of sample of cities from these metrics. An alternative approach, called *Partition Statistics*, is to divide each case of FTSP and fractal order into subsegments, and to extract metrics from each subsegment.

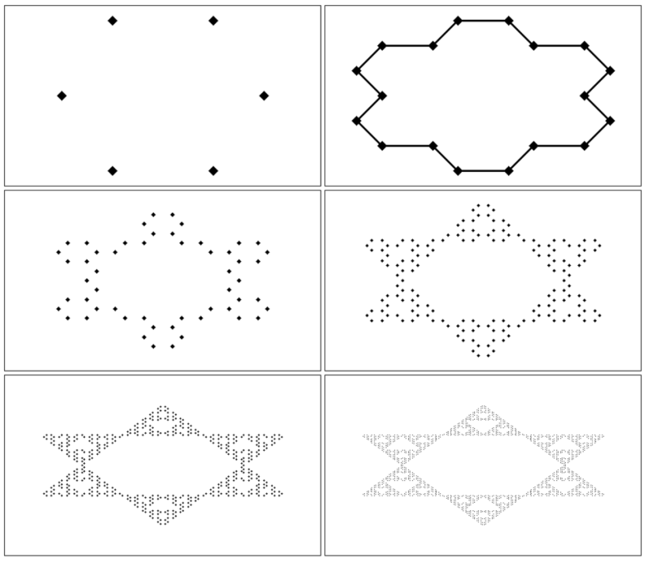


Figure 1: David Tour Coordinates - Orders 0 to 5

The instances of the problem are the metrics extracted from each subsegment. We will also test a variation of Partition Statistics labelled *Scrambled statistics* where we introduce some noise into the data set. We use GAssist [1, 12, 11] as it has been shown to be robust across a wide range of difficult data mining problems. We have used the parameters and version of GAssist specified in [2].

3.1 Global Statistics

We propose next a simple set of features that is used to characterise instances of the FTSP. The goal is to be able to decide to which fractal instance a *sampled set of cities* belongs to by looking at these features. A successful implementation of this decision step would allow to use (parts of an) instance as an *index* into the heuristic that solves it better (in some cases optimally) as shown in [7].

Please note that each family includes an infinite number of possible instances (orders), and we would like to identify a FTSP class by the value of its attributes, despite which order it represents. This means that GAssist must capture the similarities between, e.g., the first and third orders of Koch Tour, and their differences in relation to, e.g., the fifth order of MPeano. We consider 4 classes (i.e. MNPeano, NPeano, David Tour, Koch Tour), and for each class be produce instances up to order 6. A total of 24 instances are thus available for this experiment.

3.1.1 Characterization of the learning problem

We define next the set of features we use to characterise instances of the MPeano, MNPeano, David Tour and Koch Tour family of instances. We have characterized each FTSP class by 6 different continuous attributes: number of cities in the sample, maximum X and Y components, Average spread from X and Y axis center and average of nearest neighbors.

Since this model aims to avoid absolute references to the position of the coordinates in the graph, they must first undergo a normalization process. The curve is considered as if meant to fit a 1x1 square, fixed to its lower left border; this means that the lowest **x** and **y** value are 0, but the highest values need not be exactly 1 (i.e. when the figure described

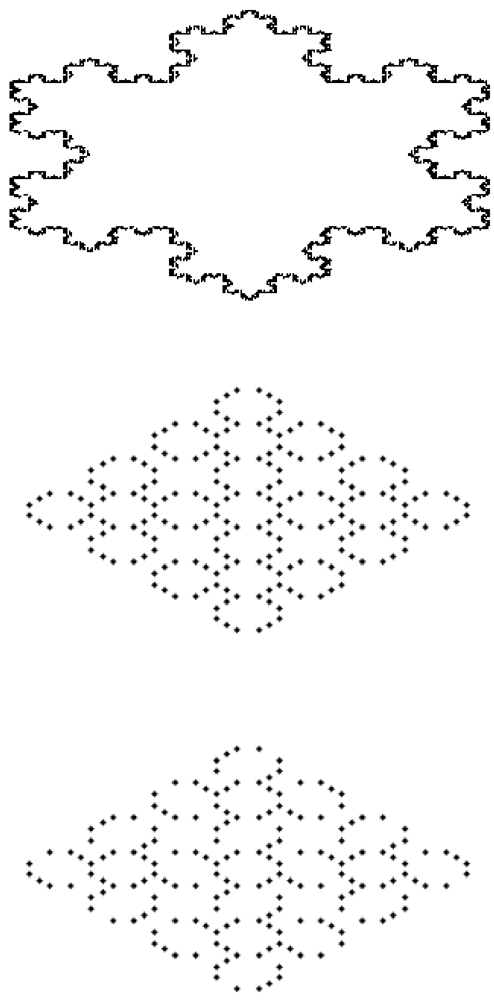


Figure 2: Order 5 of Koch, MPeano and MNPeano tours

by the curve is not symmetrical from its center). Given these constraints, the maximum X/Y component attributes are computed as follows:

$$MaximumXComponent = \frac{X_{max} - X_{min}}{max(X_{max} - X_{min}, Y_{max} - Y_{min})}$$

$$MaximumYComponent = \frac{Y_{max} - Y_{min}}{max(X_{max} - X_{min}, Y_{max} - Y_{min})}$$

The average spread from X/Y axis center attributes are intended to express a general *signature* of the shape of the curve in a 2-dimensional space. The **Spread** of a coordinate *c* from a given axis center computes "how far" the corresponding component is. The metrics are defined as follows:

$$Spread\ from\ X\ axis\ center(c) = 1 - \frac{|c_x - center_x|}{|X_{max} - center_x|}$$

$$Spread\ from\ Y\ axis\ center(c) = 1 - \frac{|c_y - center_y|}{|Y_{max} - center_y|}$$

Class	Order	Avg	St. Dev.	Avg. w/o MPeano	St. Dev. w/o MPeano	Avg. w/o MNPeano	St. Dev. w/o MNPeano
MPeano	1	70	0.48			90	0.32
MPeano	2	50	0.53			100	0.00
MPeano	3	40	0.52			90	0.32
MPeano	4	70	0.48			100	0.00
MPeano	5	0	0.00			100	0.00
MPeano	6	20	0.42			90	0.32
MNPeano	1	80	0.42	100	0.00		
MNPeano	2	0	0.00	60	0.52		
MNPeano	3	60	0.52	100	0.00		
MNPeano	4	0	0.00	100	0.00		
MNPeano	5	50	0.53	100	0.00		
MNPeano	6	90	0.32	100	0.00		
Koch Tour	1	70	0.48	30	0.48	40	0.52
Koch Tour	2	100	0.00	100	0.00	100	0.00
Koch Tour	3	100	0.00	100	0.00	100	0.00
Koch Tour	4	100	0.00	100	0.00	100	0.00
Koch Tour	5	100	0.00	100	0.00	100	0.00
Koch Tour	6	100	0.00	100	0.00	100	0.00
David Tour	1	10	0.32	0	0.00	10	0.32
David Tour	2	40	0.52	80	0.42	70	0.48
David Tour	3	100	0.00	90	0.32	100	0.00
David Tour	4	90	0.32	90	0.32	90	0.32
David Tour	5	100	0.00	100	0.00	100	0.00
David Tour	6	100	0.00	100	0.00	90	0.32

Table 1: Global Statistics Test Accuracy

The resulting value lies between 0 (when the component is on a border) and 1 (when the component is in the center).

Finally, the average of nearest neighbors attribute is the averaged value amongst all the coordinates in the curve of the nearest neighbors. The **nearest neighbors** of a coordinate is a number indicating the amount of coordinates in the curve that are at minimum distance.

3.1.2 Experimental results

In this approach we only have one instance for each FTSP class and order (24 in total). Given this small set of instances, we have chosen a *leave-one-out* validation process: We have used 23 instances for training and the 24th one for test, repeating this process 24 times, each time using a different instance for test. Also, each training process is repeated 10 times with different initial random seeds of GAssist. We will show then, for each of the 24 instances, the average accuracy obtained when the instance was used for test.

The results obtained for this model (first accuracy column in Table 1) show good results for predicting Koch Tours and David Tour with increasing orders. This can be understood as the fractal graphs are smaller on the first orders, and that means a greater proximity between the opposite sides of the obtained figure. In regard to MPeano and MNPeano tours, results show a much lower accuracy. This, however, was expected since both curves share almost all cities, and even a human observer can not distinguish them.

In order to prove our hypothesis, the experiment was rerun twice, first discarding the MPeano instances, and then without the MNPeano ones, so as to avoid confusing GAssist. The second and third accuracy columns in Table 1 confirm the hypothesis, while also maintaining the high accuracy for Koch Tour and David Tour.

3.2 Partition Statistics

The Global Statistics model succeeded at classifying a TSP instance by obtaining the attributes computed from the coordinates of the corresponding graph. With the objective of confirming the robustness of such attributes, we look now into the case when not all of the coordinates from a curve are present, but only a subset of the cities is available for deciding which heuristic to use. A **Partition** is a subset of the coordinates of consecutive cities from a given instance.

The **Partition Statistics** model implies partitioning each FTSP (and fractal order) and using these partitions to create the instances that will be fed into GAssist, using the same attributes that we proposed for the global statistics model except for the number of cities. It will not be used because, due to the way in which the FTSP are partitioned, not even all the partitions for the same order of the same TSP type share the same value for this attribute.

3.2.1 *Partition Construction*

Depending on the amount of coordinates we put in each partition, the number of partitions for each curve will vary. Likewise, if we are to establish a fixed amount of partitions to be obtained from each curve, a way has to be devised to determine the amount of coordinates in each partition. When the number of coordinates in a curve is not divisible by the number of partitions we intend to have, Algorithm 1 is applied to our model.

```
Require: partitionsLeft: Integer
  partitions = an empty collection
  CoordinatesLeft = curve.coordinates.size
  from = 1
  to = 0
  while $partitionsLeft > 0$ do
    coordinatesInPartition = Round($\frac{CoordinatesLeft}{partitionsLeft}$)
    to = min(to + coordinatesInPartition,curve.coordinates.size)
    partitionCoordinates = curve.coordinates.copy( from , to )
    partitions.add( partitionCoordinates )
    from = to + 1
    partitionsLeft = partitionsLeft - 1
  end while
  return partitions
```

Algorithm 1: Instance Partitioning

3.2.2 *Experimental Results*

We use next a 10-fold cross-validation evaluation process. In order to analyze the usefulness of the presented model in classifying partitions of the FTSP instances, the algorithms presented were used to generate data for 2 to 10 partitions, considering all 4 curves from order 1 to 6. The previous experiments were rerun, this time considering only 3 curves: David Tour, Koch Tour and MPeano. The deletion of MNPeano from the data has a simple objective: prove that the low accuracy obtained in the previous experiments is due to the similarities between MPeano and MNPeano, which is thought to confuse GAssist. Based on the idea that the higher orders provide a more distinct shape of the curves, the same experiments were rerun but omitting the first order of the curves, thus reducing the amount of available instances on one side, but allowing for a higher level of partitioning on the other, which allowed the tests to go as far as 20 partitions, leading to the possibility of a more significant analysis. Figure 3 presents a comparative of the obtained results. We can see how the accuracy given by GAssist decreases with increasing number of partitions, as well as, as expected, that higher accuracy can be obtained with three types of FTSP instead of four.

3.3 Scrambled Statistics

The *Scrambled Statistics* model objective is to distort the position of the coordinates, and check how much noise can

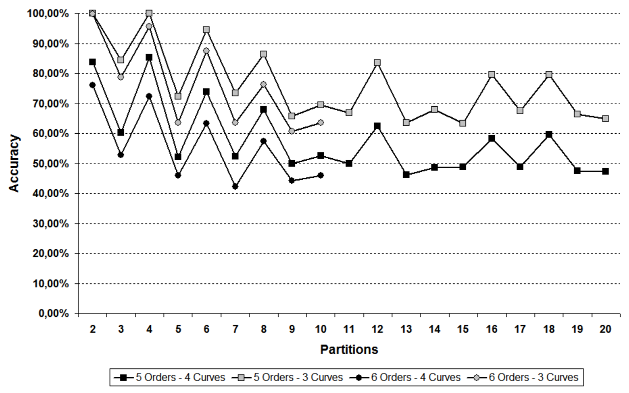


Figure 3: Partitions Model Accuracy

GAssist withstand before losing the high accuracies obtained so far. We aim then at proposing a way to add noise to the position of the coordinates, and then proceed to suggest how to experiment with the amount of noise present and understand its effect on the features of the model: we achieve this aim by computing the variance of the coordinates position in both axis. We obtain a Gaussian distribution based on it, with a mean of 0, and then we modify the coordinates positions by applying white noise using Algorithm 2.

```
Require: scramblingRatio: Float
  coordinate = original coordinate from the graph
  varianceX = variance in the original graph for the x axis
  varianceY = variance in the original graph for the x axis
  normalGaussianX = random gaussian (mean 0, standard deviation 1)
  normalGaussianY = another random gaussian
  adjustedGaussianX = 0 + squareRoot(varianceX) * normalGaussianX
  adjustedGaussianY = 0 + squareRoot(varianceY) * normalGaussianY
  adjustmentX = adjustedGaussianX * scramblingRatio
  adjustmentY = adjustedGaussianY * scramblingRatio
  scrambledCoordinate = ( coordinate.x + adjustmentX , coordinate.y + adjustmentY )
  return scrambledCoordinate
```

Algorithm 2: Coordinate Scrambling

At 20% scrambling the graph is distorted enough to lose almost all resemblance to its original shape, and this is the reason that leads us to considering 20% an the upper limit of GAssist's learning capacity in this noisy domain. It must be noted that since the idea of this model is to represent the real world problems that our proposals might find, we scramble only the testing sets, since we can ensure by construction the correct positioning of our FTSP coordinates.

3.3.1 *Experimental Results*

Experiments will analyze scrambling from 0% to 20% for two well behaving experiments from the Partition Statistics model: 8 and 4 partitions, without MNPeano, considering orders 2 to 6. Results are shown in Figure 4. Results are similar to the ones described in the previous section, although showing slightly lower accuracy, due to the introduced noise.

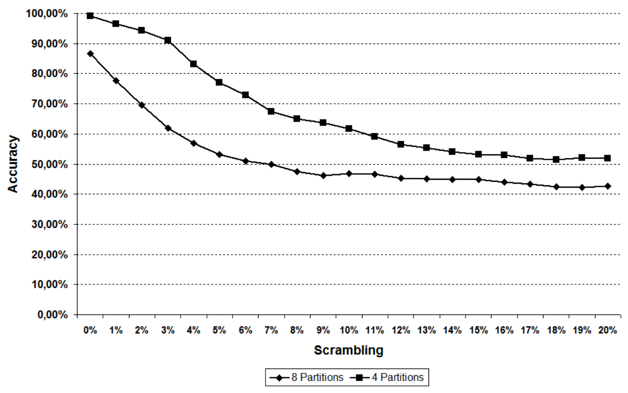


Figure 4: Scrambling Statistics Accuracy Results

4 Identifying edges from the optimal TSP solution

In this section we use GAssist to learn how to identify edges that belong to the optimal tour of an FTSP instance. Since we know by construction the optimal path of the studied FTSPs, we can easily create a dataset mixing these edges with random edges not included in the solution, thus obtaining an important group of samples to train the system.

4.1 Triplets Model

In the **Triplets** model, each instance represents a part of a path between the nodes of a TSP. Because this model was developed for learning purposes only, it was decided that instances would contain a path of size 2, that is, two edges. Also, in order to keep the ideas presented as simple as possible, we will purposefully omit any normalization process of the coordinates. So, if for example our TSP had 3 cities in coordinates (1,1),(1,2) and (2,1), a possible solution would be: $(1,1) \rightarrow (1,2) \rightarrow (2,1) \rightarrow (1,1)$. Also, the tour can be seen backwards and, to complete the possible attributes of the model, we must consider what happens when the initial vertex changes. This of course means that redundancy will exist between instances, but as we will show later, this is part of the motivation behind the proposed structure.

The objective now is to generate a group of instances large enough to be used as training and testing sets in GAssist. First we select a FTSP and its order. Then the associated instances are built using the edges for the attributes and indicating "Included" (in an optimum tour) as their class. An equal number of instances are generated considering edges that do not belong to the TSP, with class "Not Included" (in any optimum tour).

To transform the vertices into GAssist instances, we apply the following procedure. The destination vertex of the first edge and the origin vertex of the second edge are the same. To reduce the size of the instance we intend to present GAssist with the three different vertices in the right order. Six continuous attributes are used to describe these instances: **Previous X, Previous Y, Current X, Current Y, Next X** and **Next Y**.

To generate the instances of the “Not included” class we randomly choose vertices and combine them so as to generate different instances. Since we know the optimal path, we can assure that the randomly generated instances do not conform to a cycle with lesser cost. Also, since each instance is randomly generated without connecting it to the previous ones, the set is not even required to form a Hamiltonian cycle. In order to build a more interesting set, an additional condition is imposed on these randomly generated instances: the three coordinates informed must not be repeated in the same instance, thus avoiding the creation of samples that would clearly be discarded with a simple optimization of any program intending to attain the same objective of the Triplets model.

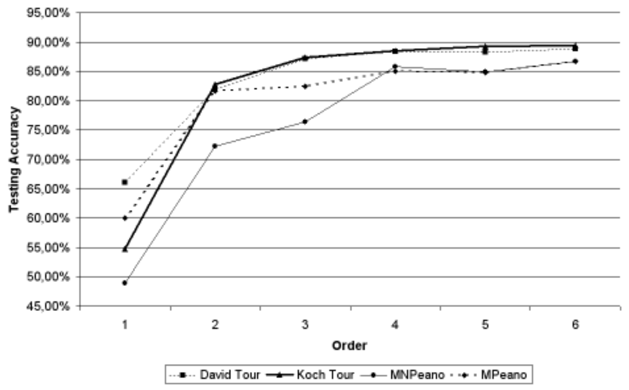


Figure 5: Triplets Model - Testing Accuracy

4.2 Experimental Results

The test accuracy averaged over 10 runs of GAssist for a fold (which belong to a particular order of a particular curve), is then averaged for all 4 folds, generating the accuracy values displayed in figure 5. Additionally, it must be pointed out that the implementation of GAssist informs the best accuracies obtained during its internal iterations for a given run; this must be taken into consideration when analyzing the meaning of the values obtained and presented.

An important feature to be remembered in future sections is that each curve presents a different starting point in the chart. Eventually, as it can be seen, all curves improve their accuracy as the order increases, although an asymptote seems to exist at 90% accuracy, impending a more satisfactory result. Despite that, GAssist has, with the approach presented in this model to represent the information, surpassed by far the 50% that a lazy classifier would have obtained (since half of the instances presented are included in the optimal tour, while half are not).

A human classifier, put to the test to do the same task, would probably obtain similar results, as no information from one used instance can determine the class of another. This the main idea that will guide the analysis and constructions of the next model: the search for features that allow the identification of a curve with its class.

5 Generating the optimal TSP path

In the previous sections we have shown how we can employ LCSs to identify individual FTSP classes as well as identifying subsegments of a path that belong to the optimal path. Now we would like to integrate all these predictive models into an heuristic that is able to solve the TSP problem. The first step, described in this section, is an heuristic that can automatically generate a close-to-optimal TSP path for each individual FTSP class.

The *Good Enough Solution Finder* is mainly based on iterating over all of the edges of the complete graph described by the coordinates it receives, and keeping only the ones allowed by the rule sets generated by GAssist from the triplets model dataset, described in section 4.1. The algorithm ideally ends up generating a Hamiltonian cycle. The algorithm is described as Algorithm 3:

```
Require: $\mathbf{C}$: a set of coordinates , $\mathbf{R}$: a rule set that indicates whether a given triplet is included or not in the solution to a TSP instance
  $\mathbf{S} = \emptyset$ {This set of triplets can be interpreted as the edges that make up the solution to the TSP.}
  for $\mathbf{c} \in \mathbf{C}$ do
    if $grade_{\mathbf{S}}(\mathbf{c}) < 2$ then
      if $grade_{\mathbf{S}}(\mathbf{c}) = 1$ then
        $\mathbf{e_1}$ = The edge $\in \mathbf{S}$ connecting $\mathbf{c}$
      else
        $\mathbf{e_1}$ = A randomly chosen edge between $\mathbf{c}$ and another coordinate $\mathbf{c}'$ with $grade_{\mathbf{S}}(\mathbf{c}') < 2$, such that it does not imply a non Hamiltonian cycle
      end if
      $\mathbf{e_2}$ = A randomly chosen edge $\neq \mathbf{e_1}$, between $\mathbf{c}$ and another coordinate $\mathbf{c}''$ with $grade_{\mathbf{S}}(\mathbf{c}'') < 2$, such that it does not imply a non Hamiltonian cycle
      if $\mathbf{R}$ classifies the triplet defined by $(\mathbf{e_1}, \mathbf{e_2})$ as included then
        $\mathbf{S} = \mathbf{S} \cup \{(\mathbf{e_1}, \mathbf{e_2})\}$
      end if
    end if
  end for
  return $\mathbf{S}$
```

Algorithm 3: Pseudocode for the Good Enough Solution Finder

Although this algorithm does not ensure optimality, it provides an interesting method to obtain quick solutions for an instance of the TSP that belongs to the class of instances upon which GAssist was trained. As this work represents a proof of concept on how to link Genetic Based Machine Learning to optimisation, we are not interested in comparing the quality of the solution generated by Algorithm 3 but rather in assessing whether (1) learning took place and (2) if it did, how it biases random tour construction. In order to do this, however, we must first find a way to choose the right rule set. This topic is covered in the next section.

6 Choosing the correct FTSP class

Just as the rules from Section 4.1 are used within Algorithm 3, we will make use now of the rule sets obtained in Section 3.1. As these rule sets classify a TSP instance into the different classes, we will present an algorithm to take advantage of this and tell us which TSP instance (MPeano/MNPeano, Koch Tour, David Tour) a set of coordinates belongs to, regardless of the order of such instance.

The algorithm to which we will refer as *Instance Classifier* requires just a few lines of code as shown in Algorithm 4.

Once the rule sets are obtained, the computation and the complexity required to classify the set of coordinates received is marginal, as can be seen in Algorithm 4.

```
Require: $\mathbf{C}$: a set of coordinates , $\mathbf{R}$: a rule set that classifies a set of Global Statistics features into one of the TSP instances
  $\mathbf{F}$ = Global Statistics features as described in Section 3.1 computed from $\mathbf{C}$
  $\mathbf{I}$ = The classification returned by $\mathbf{R}$ when applied to $\mathbf{F}$
  return $\mathbf{I}$
```

Algorithm 4: Pseudocode for the Instance Classifier

7 A system to solve TSP

Just by using the ideas proposed in Section 6 we can go from a set of coordinates to the name of the TSP instance. By means of the results presented in [8] and [7], this alone means that we can apply the optimum algorithm to solve MPeano, MNPeano and Koch Tour. Considering that some of the rule sets obtained in the experiments detailed in Section 3.1 reached a 100% accuracy, if presented with a set of coordinates belonging to any of the analyzed orders of MPeano, MNPeano or Koch Tour, we could provide the best solution to the TSP in linear time with complete certainty.

Despite these meaningful possibilities, the algorithms presented and the rule sets obtained allow us to reach even further in our ambition of solving the TSP, and so we present next a system that:

1. Takes a set of coordinates from any order and FTSP type
2. Determines with Algorithm 4 the FTSP type
3. Uses a pre-built table to deduce from the number of coordinates the order of the TSP instance
4. Applies Algorithm 3 to identify the edges that belong to a Hamiltonian Cycle, as an approximation of the optimum solution to the TSP for the given coordinates

A system as described is then equipped to adjust itself internally to produce the best possible results for the problem received as input, instead of pretending to contain a general solution for any input. Given that the "No Free Lunch Theorems" [13] warn us not to try to obtain a general solution, we believe that a system with the structure proposed is guided towards avoiding the restriction stated by these theorems, as it wraps a group of subsystems each designed for a specifies subset of the input space.

7.1 Experiment Configuration

The basics of the system implemented are those described above. In order to analyze the effectiveness of our proposal, the following experiments were prepared:

- **Input**: David Tour, Koch Tour and MPeano, from orders 1 to 4. Total inputs: 12.
- **Instance Classification Rule Set**: The rule set shown in Figure 6 was applied, since it was one of the many which got a 100% testing accuracy when it was created for Section 3.1.
- **Coordinates to order table**: Based on the instances considered, Table 2 was built into the system.
- Triplet Classifying Rule Set: In order to test the learning achieved by the rules created in Section 4.1, three different classification schemes were considered:

Instance	# of Coordinates	**Order**
David Tour	18	**1**
David Tour	54	**2**
David Tour	162	**3**
David Tour	486	**4**
Koch Tour	12	**1**
Koch Tour	48	**2**
Koch Tour	192	**3**
Koch Tour	768	**4**
MPeano	12	**1**
MPeano	28	**2**
MPeano	52	**3**
MPeano	108	**4**

Table 2: Instance order based on N^o of coordinates

1. **Random Coin**: Instead of applying a rule set, we classify a Triplet as "Included" or "Not Included" with a 50% random probability.
2. **Highest Accuracy Rule Set**: We pick, from the rule sets created in Section 4.1, that with the highest training accuracy for each instance.
3. **Ensemble of High Accuracy Rule Sets**: Considering the five rule sets with the highest training accuracy, we perform a majority voting to decide whether a Triplet should be considered or not for the solution proposed, as suggested in [3]. This includes the rule set from the previous scheme.

1. **If** $yAxisCenterAveragedSpread > 0.594 \rightarrow MPeano$
2. **If** $xAxisCenterAveragedSpread \in [0.393, 0.452] \rightarrow KochTour$
3. **If** $maximumXcomponent > 0.985 \rightarrow MPeano$
4. **Everything else** $\rightarrow DavidTour$

Figure 6: GAssist rule set used to classify the instances

In case that some coordinates were left unconnected because of a massive classification of the Triplets as "Not included", the system completes the solution by joining any such coordinates with connections to the nearest unconnected neighbour.

7.2 Experimental Results

Figures 7, 8 and 9 present the results of executing the system proposed with the configuration previously detailed. Below every image, the length of the solution is indicated.

These results show that the difference between the tour length obtained by the random coin classification and the one obtained with the rule sets increases as the number of coordinates grow. This means that as the problem to solve becomes more complex, the proposed system improves its performance over a random solution. Considering the simple structure of the algorithms used and the basic strategies applied to obtain the rule sets, the study presented in this paper leads us to think that a more complex algorithm coupled with better features for GAssist can guide the creation of high performing solutions for the TSP.

8 Conclusions and further work

In this work we presented a proof-of-concept of the use of a Learning Classifier System, GAssist, for the classification of TSP instances. We have used fractal TSP instances as these have been indexed with the heuristics that can solve them to optimality. Thus, being able to correctly classify a subset of cities into one of four fractal TSP families, namely MNPeano, MPeano, David Tour, Koch Tour, allows us to decide which heuristic to use for solving them on-the-fly. In principle it should be possible to extend this work to other fractal instances and other heuristics and even, perhaps, to non fractal TSP instances. We have also shown that a LCS might be able to learn to classify the edges of the complete graph belonging to a fractal instance into those that belong to the optimal tour and those that do not. In contrast to similar work done with neural network where the NN is used to learn the optimal tour, the LCS provides *insights* on why it chooses one edge over another to be included in the optimal tour as its rules are human-readable and explicit.

Future work will include a more systematic analysis of the work done here by (1) substantially extending the order for which instances have been generated, (2) systematically investigating richer representations for the various data mining problems and (3) testing smarter ensemble techniques.

9 Acknowledgments

NK acknowledges the EPSRC for grant EP/D061571/1.

10 References

[1] J. Bacardit. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain, 2004.

[2] J. Bacardit. Analysis of the initialization stage of a pittsburgh approach learning classifier system. In *GECCO 2005: Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1843–1850. ACM Press, 2005.

[3] J. Bacardit and N. Krasnogor. Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In *Proceedings of the 9th International Workshop on Learning Classifier Systems.* (to appear), LNAI, Springer-Verlag, 2008.

[4] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. Hayes-Roth and F. Waterman, editors, *Pattern-directed Inference Systems*, pages 313–329. Academic Press, New York, 1978.

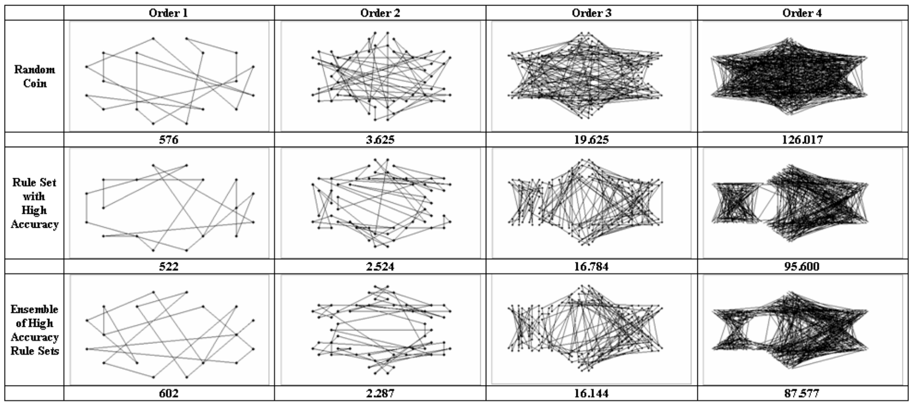
[5] A. Mariano, P. Moscato, and M. Norman. Arbitrarily large planar etsp instances with known optimal tours. *Pesquisa Operacional*, 15:89–96, 1995.

[6] A. Mariano, P. Moscato, and M. Norman. Using l-systems to generate arbitrarily large instances of the euclidean traveling salesman problem with known optimal tours. In *Anales del XXVII Simposio Brasileiro de Pesquisa Operacional*, Vitoria, Brazil, 6-8 Nov. 1995.

[7] P. Moscato and M. Norman. An analysis of the performance of traveling salesman heuristics on infinite-size fractal instances in the euclidean plane. Technical report, CeTAD - Universidad Nacional de La Plata, 1994.

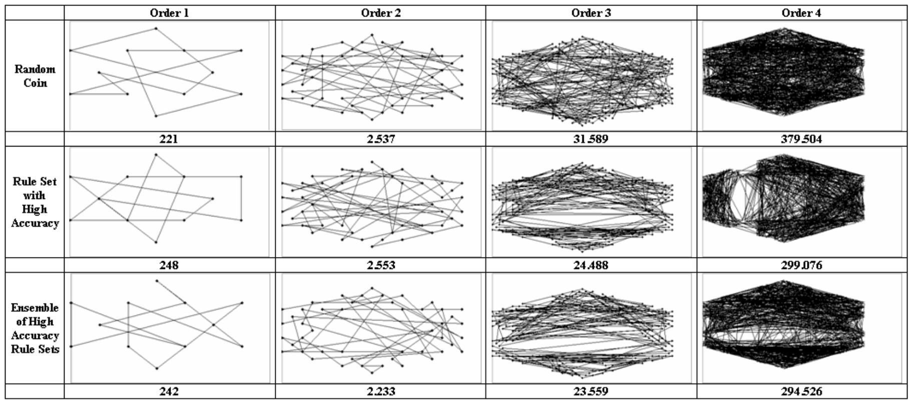
[8] M. Norman and P. Moscato. The euclidean traveling salesman problem and a space-filling curve. *Chaos, Solitons and Fractals*, 6:389–397, 1995.

[9] H. Peitgen, H. Jürgens, and D. Saupe. *Chaos and Fractals*. Springer, February 2004.



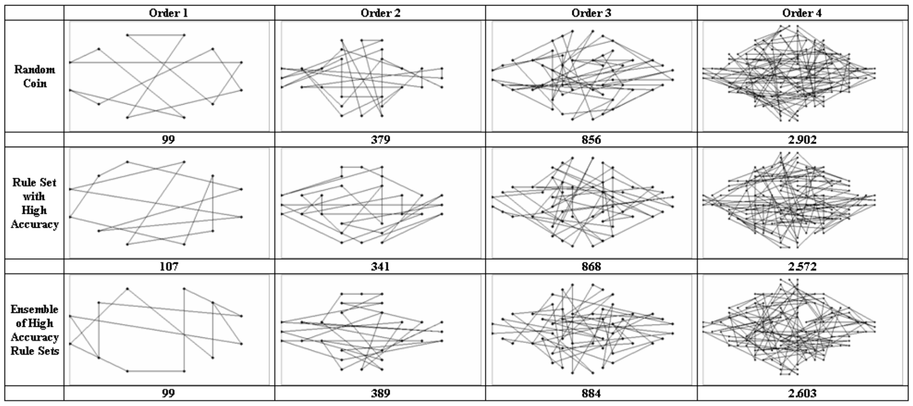
	Order 1	Order 2	Order 3	Order 4
Random Coin	576	3.625	19.625	126.017
Rule Set with High Accuracy	522	2.524	16.784	95.600
Ensemble of High Accuracy Rule Sets	602	2.287	16.144	87.577

Figure 7: David Tour results with the TSP Solving System for different rule sets



	Order 1	Order 2	Order 3	Order 4
Random Coin	221	2.537	31.589	379.504
Rule Set with High Accuracy	248	2.553	24.488	299.076
Ensemble of High Accuracy Rule Sets	242	2.233	23.559	294.526

Figure 8: Koch Tour results with the TSP Solving System for different rule sets



	Order 1	Order 2	Order 3	Order 4
Random Coin	99	379	856	2.902
Rule Set with High Accuracy	107	341	868	2.572
Ensemble of High Accuracy Rule Sets	99	389	884	2.603

Figure 9: MPeano results with the TSP Solving System for different rule sets

[10] S. Smith. *A Learning System Based on Genetic Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[11] M. Stout, J. Bacardit, J. D. Hirst, and N. Krasnogor. Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics*, In press, 2008.

[12] M. Stout, J. Bacardit, J. D. Hirst, R. E. Smith, and N. Krasnogor. Prediction of topological contacts in proteins using learning classifier systems. *Soft Computing, Special Issue on Evolutionary and Metaheuristic-based Data Mining (EMBDM)*, In Press, 2008.

[13] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.