

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis de Licenciatura

**Learning Classifier Systems for
optimisation problems: a case
study on Fractal Travelling
Salesman Problem**

by

Maximiliano Tabacman

Supervisors: Natalio Krasnogor & Irene Loiseau

Buenos Aires, 2008

Abstract

The Traveling Salesman Problem implies finding a Hamiltonian cycle of minimum cost in a fully connected graph with costs associated to its edges. Although mathematically simple to describe, the TSP belongs to the group of NP-Hard computational problems. This means that exact solutions are costly in time and resources, and real world problems associated with it would be greatly benefited by faster and better approaches to solving it.

In this thesis, we propose the use of a Learning Classifier System. Since an LCS requires information presented in the form of valued attributes and a corresponding class, we present a way of converting instances of TSP to such valued attributes, taking advantage of the group of TSP instances known as Fractal TSP.

After dealing with varied conversion and experimentation models, it can be seen that with the right parameters and structure, this Machine Learning technique not only obtains highly accurate results but also delivers a human understandable explanation of the rules discovered. Apart from being able to classify with no chance of error the possible inputs, we present a system that uses the results obtained, serving as a proof of concept that shows a way to solve Optimization Problems using Learning Classifier Systems.

Resumen

El Problema del Viajante de Comercio (en inglés *Travelling Salesman Problem* ó *TSP*)[21] implica encontrar un circuito Hamiltoniano de costo mínimo en un grafo completo K_n . Aunque su descripción a nivel matemático es simple, resolverlo es extremadamente costoso, ya que es uno de los problemas que se conocen como NP-Hard. Esto nos lleva a la necesidad de buscar mejores maneras de encararlo, para reducir los tiempos de procesamiento y asegurar mejores resultados.

En esta tesis se propone el uso de sistemas de clasificación (en inglés *Learning Classifier Systems* ó *LCS*)[13]. Como un LCS recibe información en forma de atributos y una clase asociada, presentamos una manera de convertir instancias de TSP en dichos atributos, aprovechando un conjunto particular de TSP conocido como TSP Fractales.

Luego de analizar diversos modelos de conversión y experimentación, podemos concluir que a partir de los parámetros y estructura adecuada, esta técnica de Machine Learning no sólo obtiene resultados precisos sino también de fácil lectura, en la forma de conjuntos de reglas. Además de conseguir clasificar todas las entradas analizadas con total certeza, presentamos un sistema que aprovecha los resultados obtenidos, y constituye una prueba de concepto que muestra una manera de resolver problemas de optimización a través de sistemas de clasificación.

Resumen Extendido

El Problema del Viajante de Comercio (en inglés *Travelling Salesman Problem* ó *TSP*)[21] implica encontrar un circuito Hamiltoniano de costo mínimo en un grafo completo K_n . Aunque su descripción a nivel matemático es simple, resolverlo es extremadamente costoso, ya que es uno de los problemas que se conocen como NP-Hard. Esto nos lleva a la necesidad de buscar mejores maneras de encararlo, para reducir los tiempos de procesamiento y asegurar mejores resultados.

Esta tesis propone un enfoque distinto para resolver este tipo de problemas: el uso de sistemas de clasificación (en inglés *Learning Classifier Systems* ó *LCS*)[13]. Un LCS recibe información en forma de características que describen un objeto, y un grupo de clases posibles a las cuales se podrían asignar dichos objetos. A partir de un conjunto de elementos ya clasificados y sus características, la capacidad de estos sistemas radica en poder aprender a generalizar reglas que permitan luego reconocer la categoría a la que corresponde cualquier objeto similar.

El uso de LCS puede ser encarado para solucionar TSP usando un subconjunto particular de este problema, los *TSP Fractales*[22, 19, 23, 20]. Dichos casos pueden ser generados a partir de fórmulas basadas en gramáticas[27, 26] y, gracias a sus particulares propiedades, la solución óptima viene dada desde su construcción misma, ya que la gramática no sólo nos da los datos del problema, sino que los entrega en el orden correcto para solucionarlo.

En este trabajo analizamos 4 familias particulares de TSP Fractales: *Koch*, *David*, *MPeano* y *MNPeano*. Las últimas 2 presentan una gran similitud en su construcción y forma, lo cual servirá para entender mejor la efectividad de las soluciones propuestas. Por otra parte, algunas de estas familias pueden ser resueltas de manera óptima usando heurísticas conocidas[22] En particular tanto *Nearest Neighbour* como *Multiple Fragment* resuelven instancias de MPeano, *Multiple Fragment* resuelve las de MNPeano y *Furthest Addition From Minimal Convex Hull* puede resolver Koch.

El LCS elegido es *GAssist*[1], un sistema de aprendizaje basado en el enfoque Pittsburgh, lo cual implica que su funcionamiento básico está dado por un Algoritmo Genético. Además de tener importantes optimizaciones que lo convierten

en uno de los sistemas más completos y confiables, su mayor ventaja es la de presentar los conjuntos de reglas resultantes en un formato que es compacto y fácil de leer y comprender para una persona.

El primer objetivo de este trabajo es encontrar modelos para representar instancias de los TSP Fractales con atributos que puedan servir como datos de entrada para GAssist, así como también definir los conjuntos de clasificación posibles.

Una vez obtenidos dichos modelos, procedemos a combinarlos de manera de obtener un programa que nos permite a partir de un conjunto de coordenadas en el espacio euclídeo (las cuales representan el problema a resolver): (a) obtener una clasificación que nos indique la familia de TSP Fractales a la que más se asemeja, y (b) un orden en el cual recorrer las coordenadas (la solución para el problema).

En particular, comenzamos por describir el modelo *Triplets*. Éste implica tomar un TSP Fractal con su solución conocida, y expresarlo como los distintos caminos de tamaño 2 (es decir, 3 coordenadas / 2 ejes contiguos). Con esta información construimos un listado de todos los pares de ejes que forman parte de la solución, y los clasificamos como “Incluidos en la solución óptima”. Para completar la carga de datos, buscamos otro conjunto del mismo tamaño de pares de ejes que no sean parte de la solución y los clasificamos como “No incluidos en la solución óptima”. Al correr GAssist con este esquema, obtenemos un conjunto de reglas que nos indican (para cada tipo particular de TSP Fractal estudiado) si un par de ejes deberían ser conectados de manera contigua al armar la solución. Dicho conjunto de reglas nos es de utilidad luego, al construir el programa final.

A continuación describimos el modelo *Global Statistics*. En este caso nos interesa obtener las características que diferencian a las distintas familias de TSP Fractales, ya que identificar a un problema de esta manera permitiría aplicar la técnica más adecuada para resolverlo en cada caso. El modelo convierte las coordenadas recibidas en un conjunto de atributos, entre los que se incluye el tamaño del conjunto, el promedio de vecinos más próximos, los valores máximos en los ejes X e Y luego de normalizar los datos, y una medida de la distribución de los valores para cada eje. Las posibles clasificaciones son las familias de TSP Fractales (Koch, David, MPeano, MNPeano). Los resultados de este modelo demuestran que las características elegidas permiten distinguir entre las familias, pero hay una gran indecisión cuando debe clasificar entre MPeano y MNPeano. Ya que esto parece estar relacionado con que ambas familias tienen instancias casi idénticas, nuestros experimentos incluyen también el análisis luego de considerarlas por separado (es decir, Koch, David y MPeano por un lado, y Koch, David y MNPeano por el otro). Los experimentos con 3 familias consiguen resultados que en muchos casos llegan al 100% de efectividad en la tarea de clasificación.

Luego se detallan modelos adicionales que buscan analizar la robustez de los

atributos usados en el modelo anterior.

Primero se describe el *Partition Statistics*, que utiliza partes continuas de la solución en vez de la instancia de TSP completa. Este modelo muestra que es posible identificar a que familia pertenece una parte del problema, lo cual es interesante para ser usado en caso de que se quiera subdividir un problema TSP y resolver cada sección con el modelo que sea más conveniente. Los resultados demuestran, para el análisis con 3 familias, y cuando las particiones tienen una distribución equivalente de coordenadas, que la efectividad al clasificar se mantiene arriba del 90% con bajo número de particiones, llegando en corridas individuales a obtener un 100%.

El modelo que sigue es *Sample Statistics*, el cual mantiene la idea del anterior, pero implica partir el problema en conjuntos de puntos tomados de distintas regiones del espacio euclídeo, de manera de poder identificar la familia de TSP Fractal aún cuando no estén todos los puntos presentes. En este caso la efectividad observada es menor, si bien supera el 80% cuando se utilizan hasta 5 conjuntos.

El último modelo basado en Global Statistics es el *Scrambled Statistics*. Su objetivo es analizar cómo baja la efectividad de Partition Statistics para casos que habían obtenido un 100% originalmente, al modificar levemente la posición de los datos de prueba. Esto se logra mediante la aplicación de ruido blanco utilizando una distribución normal. Los resultados muestran que, partiendo las instancias en 4 subconjuntos, la efectividad cae progresivamente a medida que aumenta el ruido. A pesar de ello, es posible obtener arriba de 80% de éxito en la clasificación si el ruido se mantiene por debajo del 4%. Debemos destacar que una modificación excesiva de la ubicación de los puntos dejaría de implicar simplemente ruido, ya que es posible que la solución óptima cambie, por lo que consideramos que estos resultados son suficientes para demostrar la robustez de los atributos con los que hemos decidido representar las instancias.

Terminada la presentación de los modelos y sus correspondientes experimentos, presentamos la estructura para un sistema como el que nos habíamos propuesto. El mismo sirve como *prueba de concepto* para demostrar la posibilidad de resolver problemas de optimización utilizando sistemas de aprendizaje. Por un lado utilizamos las reglas resultantes de Global Statistics y clasificamos con 100% de efectividad todas las instancias de TSP Fractal consideradas. Este resultado de por sí es suficiente para aplicar la heurística más conveniente de acuerdo a los trabajos previos existentes en este área. Pero luego agregamos un paso adicional, que implica tomar las coordenadas y la clasificación calculada, para intentar sugerir una solución al TSP recibido como entrada. Para lograrlo, recorreremos los distintos ejes existentes en el grafo completo descrito por las coordenadas recibidas, y utilizamos las reglas del modelo Triplets para decidir cuáles deben ser consideradas para una solución al problema. Si bien los resultados distan

de los que obtendrían los mejores algoritmos para la solución de TSP, nuestro enfoque consigue, utilizando simplemente un conjunto de reglas, proponer en un corto tiempo una solución muy superior a la que se obtiene si la decisión de los ejes a agregar se realiza con una técnica aleatoria.

Finalmente presentamos las conclusiones de nuestro trabajo. Uno de los puntos más destacables a mencionar es que el sistema propuesto permite demostrar que *es factible utilizar sistemas de clasificación para la resolución de problemas de optimización como TSP*. Por otra parte, destacamos la capacidad del sistema para solucionar distintos tipos de familias de problemas, gracias a que utiliza técnicas de decisión que le permiten ajustar internamente el algoritmo a utilizar para la resolución de cada caso.

El trabajo se completa con una revisión de los puntos de cada modelo que pueden ser optimizados o extendidos, en trabajos futuros.

Acknowledgments

This thesis is not something I achieved on my own, but with the effort and support of a large number of people, to which I would like to thank.

First of all I thank my parents, Alfredo and Teresa, and dedicate this work to them. I thank them, for the choices they made allowed me to be the person I am today. To my sister Melisa, I thank her friendship and teachings. I thank Nai for her company and her love, and the countless readings of every little detail in this work.

I also thank my friends, which stood by my side all these years while my studies stole them of our time together. I thank my classmates (specially Z-Glib), for they taught me and worked with me term after term, so that we could reach our goals as a group.

I thank the University of Buenos Aires and its teachers, since it is them who inspires us to research and share knowledge. To all of my workmates at Mercap, I thank them, for their advice has made this thesis even better. I must also thank the people at the University of Nottingham, who shared their counsel and treated me as one of their own.

Finally, I thank my supervisors, Natalio and Irene, for their guidance and the time they invested in my work.

Contents

Abstract	c
Resumen	e
Resumen Extendido	g
Acknowledgments	k
Contents	I
1 Introduction	1
1.1 Motivation	1
1.2 Document Structure	2
2 Traveling Salesman Problem	5
2.1 Chapter Overview	5
2.2 Introduction	5
2.3 Solving Techniques	6
2.3.1 Exact Methods	6
2.3.2 Heuristics	6
2.4 Variations	7
2.4.1 Euclidean TSP	7
2.4.2 Generalized TSP	7
2.4.3 Graphical TSP	7
2.5 Fractal TSP	7
2.5.1 Definition	7
2.5.2 L-Systems	8
2.5.3 The instances	9
2.5.4 Properties	12
3 Learning Classifier Systems	21

3.1	Chapter Overview	21
3.2	Definition	21
3.3	Variations	24
3.3.1	Michigan approach	24
3.3.2	Iterative Rule Learning approach	25
3.4	GAssist	25
3.4.1	GABIL	25
3.4.2	Additional Framework Considerations	26
3.4.3	Contributions	26
3.5	Data Mining	27
4	LCS for classifying TSP optimal edges	29
4.1	Chapter overview	29
4.2	Triples	30
4.2.1	Description	30
4.2.2	Experimental Results	34
4.2.3	Learning space	35
4.2.4	Conclusion	36
4.2.5	Future Work	36
5	TSP features for instance classification	39
5.1	Chapter overview	39
5.2	Global Statistics	39
5.2.1	Description	39
5.2.2	Experimental Results	43
5.2.3	Future Work	44
6	Robustness analysis	49
6.1	Chapter overview	49
6.2	Partition Statistics	49
6.2.1	Description	49
6.2.2	Experimental Results	52
6.2.3	Conclusions	55
6.2.4	Future Work	58
6.3	Sample Statistics	59
6.3.1	Description	59
6.3.2	Experimental Results	60
6.3.3	Conclusions	60
6.4	Scrambled Statistics	65
6.4.1	Description	65
6.4.2	Experimental Results	68
6.4.3	Conclusions	69

7 From instance/edge classification to TSP optimisation: a proof of concept	71
7.1 Chapter overview	71
7.2 Solving TSP with the rules obtained for the Triplets model	71
7.2.1 Objective	71
7.2.2 Limitations	73
7.3 Identifying the curves with the rules obtained for the Global Statistics model	73
7.3.1 Objective	73
7.3.2 Algorithm description	74
7.4 A system to solve TSP	74
7.4.1 Experiment Configuration	75
7.4.2 Experimental Results	76
7.4.3 Conclusions	76
7.4.4 Future Work	77
8 Final Conclusions	81
8.1 Summary of what has been done	81
8.2 Summary of what comes next	82
8.2.1 Triplets	82
8.2.2 Statistics Features	82
8.2.3 TSP Solving System	83
Bibliography	85
List of Algorithms	89
List of Figures	90
List of Tables	92
Index	93

Chapter 1

Introduction

1.1 Motivation

An optimisation problem is one where we want to find a solution that minimizes some cost associated with it, or maximizes some measure of gain. This is referred to as the optimal or best solution.

A lot of the most studied problems still require complex techniques to be solved in reasonable time. Such is the case of the Travelling Salesman Problem, which is the one dealt with throughout this work.

In order to solve optimisation problems, exact algorithms can be used, which are costly in time and resources. A common alternative is to use heuristics, which obtain approximate solutions in polynomial time.

The main objective of this work is to present a proof of concept on how to solve an optimisation problem by means of a Learning Classifier System. These programs take as input: (a) information on a subject in the form of valued attributes, and (b) knowledge on some classification for such data. The output produced is a number of rules that allow to classify, from then on and in reasonable time, related information of the same domain into the specified classes.

Although optimisation problems and LCS do not seem to share a common objective, we believe that a link is possible between them. That is why this thesis is aimed at finding and taking advantage of such a connection, to aid in the search for optimal solutions to optimisation problems.

To the extent of our knowledge, there is no similar work to compare it to in terms of algorithms and the modeling of the problem. Despite that, some of the ideas presented are based on existing research in different fields.

In the area of TSP, we are interested in one special family, known as **Fractal TSP** (FTSP)[22, 19, 23, 20]. Since for these cases the optimum solution is known by construction, we can build any number of instances for the problem and focus

on the analysis of the model with ease.

The FTSPs are built by means of a context-free generative grammar called L-Systems[27][26]. They allow us to express an FTSP in a coded format which is useful when implementing the algorithms needed for our proposal.

As for LCS, we choose to use GAssist[1]. This system can receive the features from the input data in a number of ways (integer, real valued, nominal) and produce human understandable rules that allow a good classification. It also has some additional properties of interest, such as optimizations to improve its efficiency and response time, and automatic discretization of real valued features.

In order to reach our objective, several intermediate **models** are revised, which *map the coordinates that make up an instance of the FTSPs into attributes that can be used as input for GAssist*.

We aim at building a system 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.

In order to do that, one of the main ideas presented is a way of determining, from a group of already existing heuristics used to solve TSP, which to apply for a given problem.

Also, we build an algorithm that, although not optimally, should provide an interesting method to obtain quick solutions for an instance of the FTSP studied. Since we intend to present a proof of concept, we are not interested in comparing the quality of the solution generated but rather in assessing whether (1) learning took place and (2) if it did, how much it improves over a randomly built solution.

1.2 Document Structure

Next is a more in depth detail of the contents of this thesis, chapter by chapter:

Chapter 2 describes the **traveling salesman problem**, that is, the search for a Hamiltonian cycle of minimum cost in a fully connected graph with costs associated to its edges. After the introduction and historical review, several variations of the TSP are presented. In particular we focus on a special class of TSP instances, namely, **Fractal TSPs**. These instances are particularly appealing for benchmarking optimization and machine learning algorithms for the TSP because (a) they are non-trivial instances, (b) are well characterized in terms of which of the studied heuristics can/cannot solve them exactly, (c) can be made of an arbitrary large size, remarkably, (d) by construction the optimal tour through the set of cities in these instances is known and (e) being fractal they possess clear “patterns” that can be exploited by a learning mechanism to improve its own search process. That is, these instances allow us to reliably test the scalability of optimization and machine learning methods.

Chapter 3 revises the main concepts of **Learning Classifier Systems**. These

machine learning techniques can improve their performance at a task after being provided with training and testing data and can present conclusions in the form of rules. We delve deeper into the LCS used in this work, **GAssist**, chosen because of its human understandable outputs. This allows us to grasp the full potential of applying machine learning to optimisation problems.

The first use of GAssist to solve TSP is presented in Chapter 4. Here we introduce the **Triplets** model. With it we intend to train GAssist to recognize when a set of edges is part of the optimum solution to a TSP. Although the experiments presented take advantage of the studied Fractal TSPs, we intend to help create the knowledge to solve any instance of TSP. After the description and experimental results of the model, we mention some future work associated with it.

In Chapter 5 we move on to the **Global Statistics** model. Here we present some novel ideas about how to extract features from a TSP instance, by taking into consideration the spatial relation of the coordinates in it, that could be used to classify unseen instances or parts-of instances. By being able to classify a given instance into one of the various fractal TSP families it is possible to a priori decide which, among several available, heuristic must be applied. According to the **no free lunch theorems**[35][34], no single heuristic performs better than the rest for all problems/instances, hence being able to predict which one to use is a key milestone towards a more robust and scalable optimization. Just like in the previous chapter, we explain some details of implementation, review the experiments prepared and their results, to conclude with some observations and a few topics for future research.

Maintaining the structure and attributes of the Global Statistics model, Chapter 6 presents some variations to it. First we explain the **Partition Statistics** model, which implies partitioning the coordinates of an instance and training GAssist to recognize now not the entire graph but these subparts instead. Then the **Sample Statistics** model is presented, where the subgraphs considered are built in a random order instead of keeping the original relation between the coordinates. We then present the **Scrambled Statistics** model, which implies an analysis of the Partition Statistics model when the testing data is affected by white gaussian noise. For every model mentioned we detail its structure, experiments and results.

In Chapter 7 all of the previous ideas are merged to analyze the possibility of suggesting a solution to the TSP based on the knowledge acquired by GAssist using the models proposed. With a simple program, composed of two sub-algorithms, we aim to carry on this task. The first part of the chapter shows how to use the rule sets from the Triplets model to build a Hamiltonian Path. The second part details how to identify which of the TSP instances mentioned a graph belongs to. Finally, the third part of the chapter unifies both algorithms, producing

interesting results for several TSP instances generated from the studied cases.

Chapter 8 presents conclusions for the different models, as well as a general comment on the findings of this thesis. We close with a list of different topics for future research.

Chapter 2

Traveling Salesman Problem

2.1 Chapter Overview

This chapter begins the theoretical background of this thesis. The traveling salesman problem is first introduced, along with historical details and its formal definition. Then a group of commonly used techniques for its solving are presented, followed by some well known variations to the original problem. After this basis is established, the reader will be presented with related and also important concepts: Fractal TSPs and L-Systems. The chapter ends with a summary of its contents.

2.2 Introduction

Although it was probably first referred to in a scientific paper in 1932[21], the Traveling Salesman Problem presents a situation that is still today of interest to researchers worldwide. It has become one of the most investigated graph optimization problems, for despite its definition is simple, its solution is clearly not.

The most common references to TSP imply the definition of the symmetric traveling salesman problem.

Definition 1. *Symmetric Traveling Salesman Problem (STSP): Given a fully connected graph K_n with costs associated to its edges, find the Hamiltonian cycle of minimum cost.*

It can also be expressed in terms of a digraph, in which case it is referred to as asymmetrical traveling salesman problem.

Definition 2. Asymmetric Traveling Salesman Problem (ATSP): Given a fully connected digraph \vec{K}_n with costs associated to its arcs, find the Hamiltonian cycle of minimum cost.

The TSP belongs to the NP-hard class of problems[28].

2.3 Solving Techniques

As with any NP-hard optimization problem, investigations regarding TSP have lead to the development of both exact and approximate algorithms to solve it. Research achieved so far has been limited to algorithms that find the exact solutions for many graphs. Although every day new and bigger ones are solved, many more still remain unsolved. For these cases the attention is turned to the use of heuristics.

2.3.1 Exact Methods

Most exact methods, such as Branch and Cut algorithms, use the Dantzig-Fulkerson-Jhonson[9] integer programming model for ATSP. The objective function to minimize is $\sum_{i=1}^n \sum_{j=1}^n d_{ij}x_{ij}$ where d_{ij} represents the established weight from i to j , and x_{ij} indicates whether that arc is to be used in the proposed solution. Apart from the mentioned equation, the following constraints must hold:

1. Each vertex has exactly one incoming and one outgoing arc.
2. To avoid non-hamiltonian cycles, no proper subtour of vertices S can have a length of $|S|$ arcs.

2.3.2 Heuristics

Heuristics for the TSP can be divided into those that build up a tour trying to obtain a good solution (constructive), and those that chose an initial tour and try to improve on it (improvement).

Constructive heuristics encompass techniques that range from the simple yet inefficient *greedy algorithm* (start from a random vertex and move to the unvisited vertex with minimum cost) to the *vertex insertion* heuristic (start with a determined path in the graph and replace an arc (x, y) with $(x, z), (z, y)$ so that $weight(x, y) > weight((x, z) + weight((z, y))$).

Metaheuristics applied to the TSP are varied. Some of the most used are: genetic algorithms, tabu search and ant colony optimization.

2.4 Variations

2.4.1 Euclidean TSP

Instances of TSP where the Triangle Inequality holds are known as Euclidean TSPs. This means that the vertices can be mapped to coordinates in the euclidean plane and the weight of the edges(arcs) is define euclidean distances between them. An euclidean TSP is by definition a STSP. Although this simplification can reduce computation times and adds useful properties, it remains an NP-hard problem[24]. Existing heuristics take advantage of this case and its properties, in order to reduce computing times, and obtain approximate solutions.

2.4.2 Generalized TSP

The name of generalized TSP applies to the case where the tour to be found requires not to visit all vertices but at least one (or only one depending on the variation) from each of n clusters of vertices. In [10] a sample conversion algorithm from generalized TSP to TSP is presented. Although exact algorithms and heuristics exist for both AGTSP and SGTSP, it is also common practice to transform this cases to ATSP and STSP, in order to allow the application of the algorithms presented in previous sections.

2.4.3 Graphical TSP

The graphical variation of the TSP implies a relaxation of the constraints that allows the resulting tour to visit vertices and edges(arcs) more than once. This is related to a more general problem referred to as Steiner TSP. For more information on both topics the reader is encouraged to refer to [5].

For more on the history and variations of the traveling salesman problem, <http://www.tsp.gatech.edu/index.html> provides a rich amount of information.

2.5 Fractal TSP

2.5.1 Definition

Despite the difficulties in finding an exact solution for the TSP in general, as was previously presented some variations exist that help deal with situations where the optimal path is easier to find. This thesis focuses on one of such variations, known as **Fractal TSP** (FTSP) and well studied in [22, 19, 23, 20].

A Fractal TSP is built by using context-free generative grammars called L-Systems [27, 26] (described next in section 2.5.2). It is expressed as a sequence of coordinates, which as in any TSP can be interpreted as the positioning of cities in

the euclidean plane. But also, the sequence itself describes the optimal solution to the instance of the problem. This means that a Fractal TSP is at the same time the description of the problem (a set of coordinates) and an ordered list that explains how to solve it. To further clarify we quote from [22]:

“We turn the curves into TSP instances by inserting cities during the construction process. Often we will simply place one city at each corner of the curve. This gives us a simple mechanism for defining TSP instances of infinite size”.

“In particular we are interested in fractal TSPs of which the curve corresponds to the unique shortest tour through its set of cities”.

2.5.2 L-Systems

Before proceeding forward, L-Systems need to be introduced in order to present a way in which to formally define fractal instances of the TSP.

Definition 3. *An **L-System** is a context-free generative grammar. This means it contains the instructions to build a string, starting from an initial value and expanding by using rules. Its attributes are:*

1. An **alphabet** of possible symbols
2. A **starting symbol** composed of items in the alphabet
3. A group of **constants**, which once present in a string remain and will not be further affected by applying rules
4. A set of **production rules**, which indicate how to replace a symbol with a collection of constant and non-constant symbols

Definition 4. *The **order** of an L-System is the amount of times that the transformation rules are to be applied. The starting string represents order 0, the results after the first application are referred to as the first order, and so on.*

To provide further detail on how a particular L-System is coded, Algorithm 2.1 is presented. The pseudocode shows how to produce a string of order $n + 1$ when the string representing order n is received as input

```

input: String
output ← an empty string
FOR (each character c in input)
    expansion ← production rule for c if it exists, or c
                itself in case it is a constant
    output ← output + expansion

```

```
RETURN output
```

Algorithm 2.1: L-System Pseudocode

As an example of an L-System, the attributes in Algorithm 2.2 are proposed.

```
Begin : A
Replacement Rules :
A → AA
```

Algorithm 2.2: L-System Example

This means that the alphabet to be used is $\{A\}$, the starting symbol is A and there are no constants, since A has replacement rules defined. The first order of this system is AA, the second AAAA, etc.

For a more complex example, we present a way of generating fibonacci numbers in Algorithm 2.3.

```
Begin : A
Replacement Rules :
A → B
B → AB
```

Algorithm 2.3: Fibonacci length strings L-System

In this case the alphabet is comprised of $\{A, B\}$ and again there are no constant values. If this system is expanded, we have the results shown in Table 2.1:

Order	Result String	Length
0	A	1
1	B	1
2	AB	2
3	BAB	3
4	ABBAB	5
5	BABABBAB	8
6	ABBABBABABBAB	13
7	BABABBABABBABABBAB	21
8	ABBABBABABBABABBABABBABABBABABBAB	34

Table 2.1: Fibonacci length strings L-System. Orders 0 to 8

As will be seen next, the resulting string of the L-System has to be interpreted by a “drawer” or “coordinate builder”. Commands like **Forward** and **Rotate** $\pm X$ degrees are common, as well as other more complex instructions.

2.5.3 The instances

Because of the importance of the different fractal curves referred to in the previous section to the development of this thesis, we will review them one by one,

including a detail of its structure and its building rules. Because of a limit in time and processing power, only the first six orders of the curves will be analyzed in this work.

Koch Tour

Starting with an equilateral triangle Δ , each order replaces the lines in the previous one with a $_ \wedge _$ giving a pointed shape that reproduces this shape in each of its points, which then is reproduced on each of its points, and so on.

The L-System required to building the Koch Tour is:

```
Begin: F - - F - - F
Replacement Rules:
F → F + F - - F + F
```

Algorithm 2.4: Koch Tour L-System

where F implies drawing a line forward, + implies increasing the angle by $\frac{1}{6}$ of a turn (360/6 degrees), and - implies decreasing it by the same amount.

Figure 2.1 shows the distribution of the coordinates for Koch Tour at orders 0 to 5. It presents all of the coordinates obtained as positions in the euclidean space. Figure 2.2 shows the path connecting the coordinates according to the sequential order in which they are obtained. This means that it describes the exact solution, as obtained from the application of the L-System while building the FTSP.

David Tour

A David Tour is constructed with an iterative process described in detail in [22, 25]. This FTSP must not be confused with the Star of David fractal described in <http://library.thinkquest.org/26242/full/types/ch7.html>.

Following is a detail of the L-System needed to build David Tour instances:

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

Algorithm 2.5: David Tour L-System

where F and - keep the previously defined behavior, X is ignored for drawing purposes, and ! indicates that the angle considered is to be negated. The ! symbol implies that the drawer is constantly switching between 60 and -60 degrees, allowing us to obtain the rotating triangles needed for the figure.

In figure 2.3 the distribution of the coordinates for David Tour at orders 0 to 5 can be observed. Figure 2.4 presents, as in the previous case, the exact solution.

MPeano

The MPeano tour can be summarized as a combination of 2 MPeano/2 curves, which are in turn built based on lines inside imaginary triangles. Although this quick description may seem confusing, figures 2.5 and 2.6 provide a glance at the true complexity of this curve. In order to understand the inner calculations required for the figures and some reasoning behind them, the interested reader is referred to [22].

As with previous curves, MPeano can be described by means of an L-System, which is shown next:

```

Begin: X F F - - A F F - - X F F - - A F F
Replacement Rules:
X → +!X!FF-B@Q2F@IQ2-!X!FF+
F → <empty string>
Y → FFY
A → B@Q2F@IQ2
B → AFF

```

Algorithm 2.6: MPeano L-System

Just as before, each new curve presented includes more complex drawing rules. First it must be noted that MPeano uses an angle of $\frac{1}{8}$ of a turn, that is, $360/8$ degrees. As for the rules, we will first clarify that F's are removed when reading a string and building the next order, as can be inferred from the second rule. When @ is encountered, it indicates the drawer that the line segment (the distance to advance before placing a city when F is encountered) must be multiplied by the value following. If a number follows, multiplication is carried out normally. If Q is encountered, the square root of the value following Q is considered. If I is encountered, the value following is inverted. Some examples are in order to clarify this operators:

- @3 multiplies by 3
- @I3 multiplies by $\frac{1}{3}$ (divides by 3)
- @Q3 multiplies by $\sqrt{3}$
- @IQ3 multiplies by $\frac{1}{\sqrt{3}}$ (divides by $\sqrt{3}$)
- @QI3 multiplies by $\sqrt{\frac{1}{3}}$

MNPeano

MNPeano presents a striking similarity to MNPeano (as can be guessed from the similarity in their names). The position of the coordinates is so similar, in

fact, that with a high enough order, one can be mistaken for the other, as the only difference is that MNPeano lacks some of the cities of MPeano. As for its construction, MNPeano uses almost the same rules, but it omits a city every now and then (in a pattern detailed in [22]).

The L-System for MNPeano is explained below:

```

Begin: X F F - - A F F - - X F F - - A F F
Replacement Rules:
X → +!X!@2F@.5-B@Q2F@IQ2-!X!FF+
F → <empty string>
Y → FFY
A → B@Q2F@IQ2
B → AFF

```

Algorithm 2.7: MNPeano L-System

Figure 2.7 presents the coordinates included in MNPeano tours of order 0 to 5, while figure 2.8 adds the lines that connect them, forming an exact solution to the TSP. These solutions use the same lines that the MPeano tours shown previously.

2.5.4 Properties

Following we present a list of interesting properties that hold for all orders of the FTSPs that will be used throughout this work:

1. By construction, the optimal path (that is, the exact solution to the TSP) is known. The fact that this property is true even for instances with millions of cities makes them specially interesting for research.
2. As higher orders provide increasing number of cities, the limit on the size of the instance is only determined by the needs of the researcher, and the computing resources available. Even this is not so much because of the complexity of the building algorithm, but because of the limited capacity of computers languages to handle efficiently collections of such sizes.
3. It has been proved that some well known heuristics can solve (that is, provide an exact solution to) some of the FTSPs considered in this thesis, but none of those analyzed can solve all of them [22]. In the cited paper it is stated:

“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”.

The specific heuristics mentioned are: *Nearest Neighbour* and *Multiple Fragment* both solve MPeano instances, *Multiple Fragment* solves MNPeano and *Furthest Addition From Minimal Convex Hull* solves Koch Tour.

4. These instances have both global and local features that make them well suited for the exploratory investigations that are the objective of this thesis.

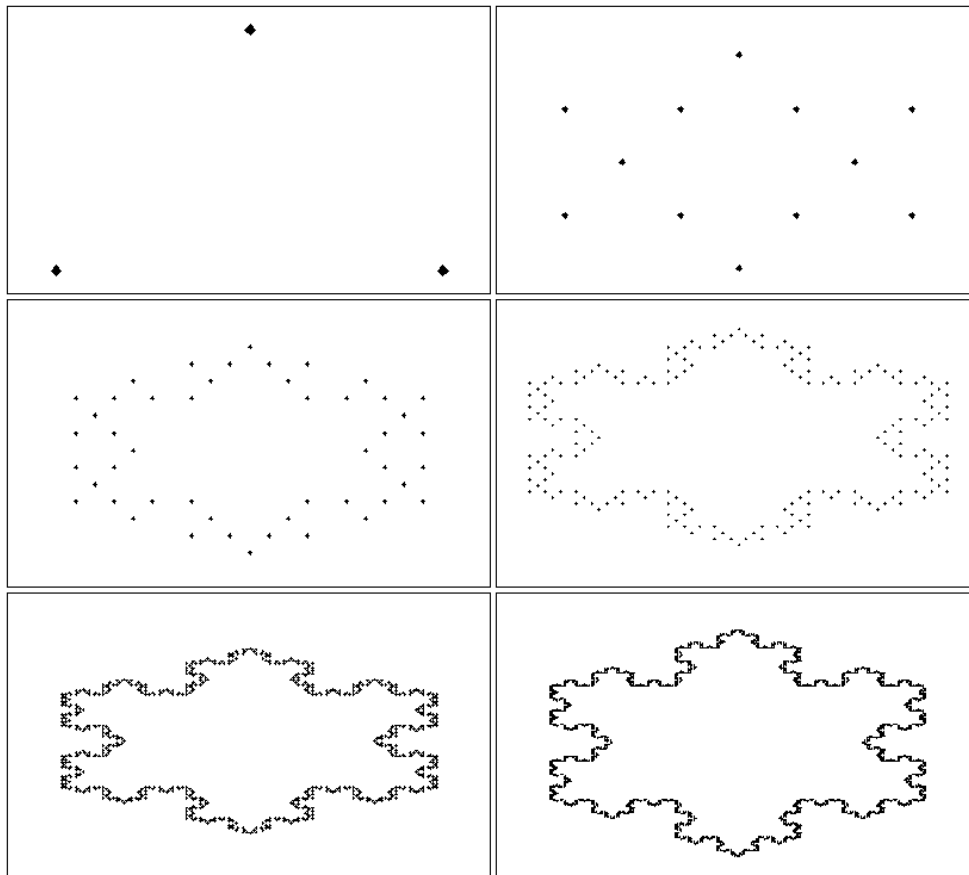


Figure 2.1: Koch Tour Coordinates - Orders 0 to 5

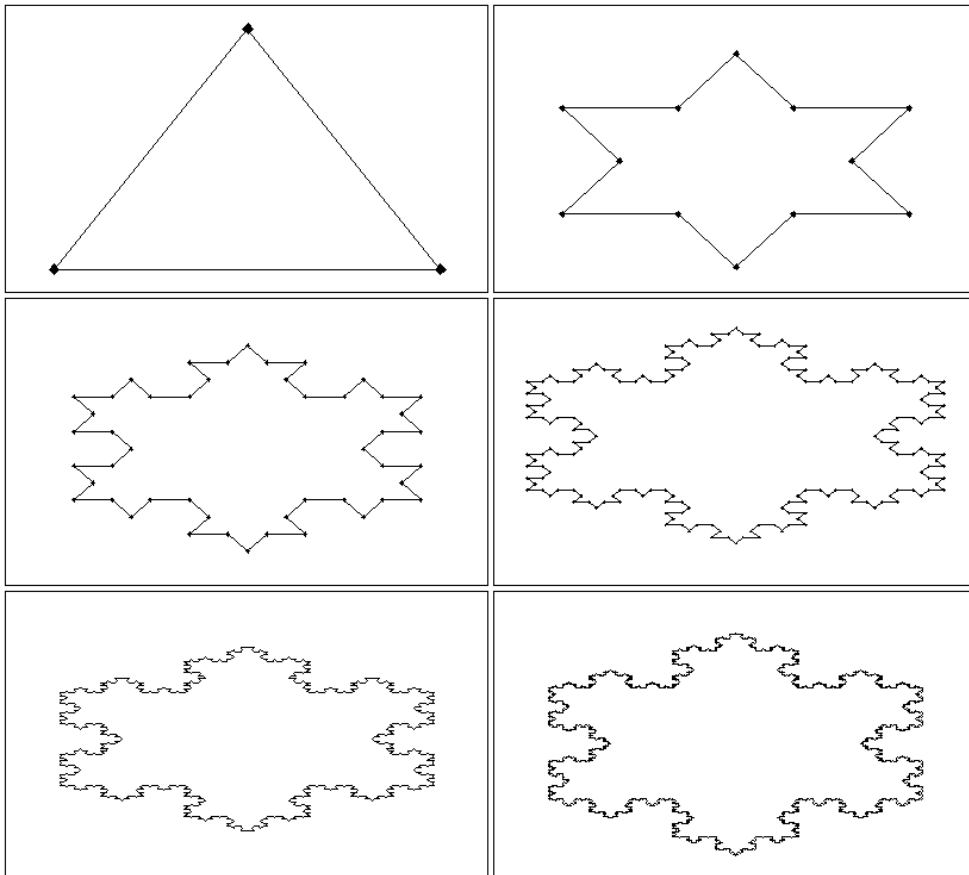


Figure 2.2: Koch Tour Exact Solutions - Orders 0 to 5

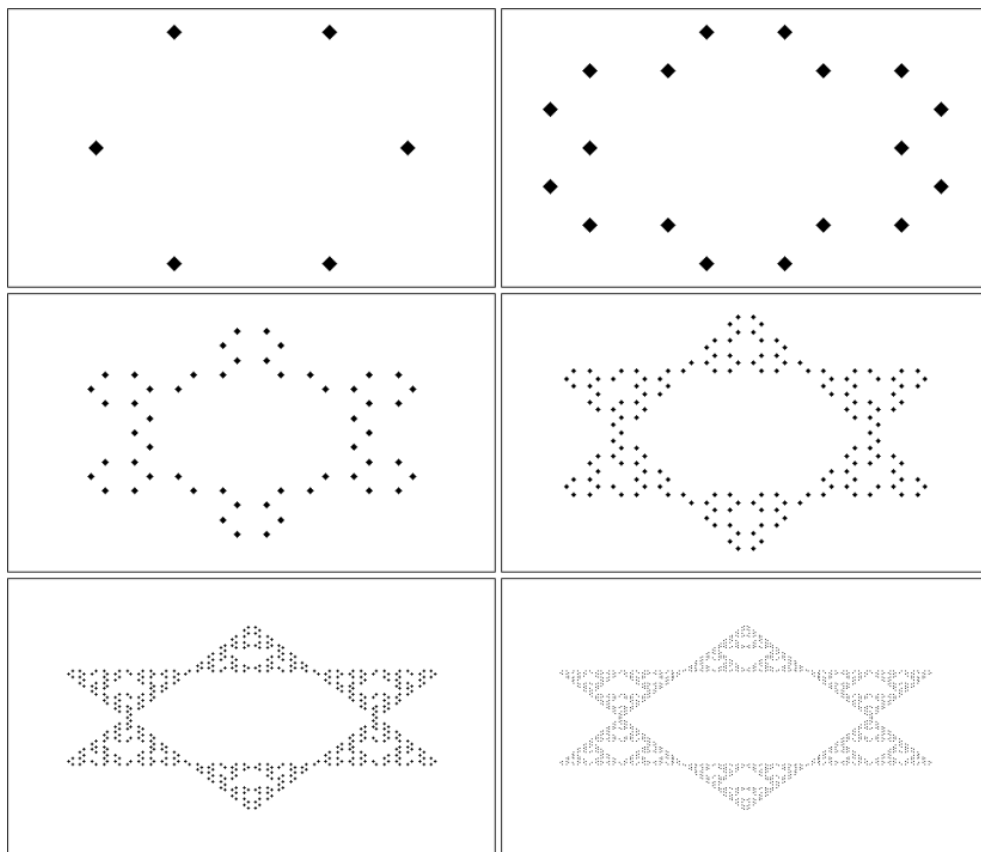


Figure 2.3: David Tour Coordinates - Orders 0 to 5

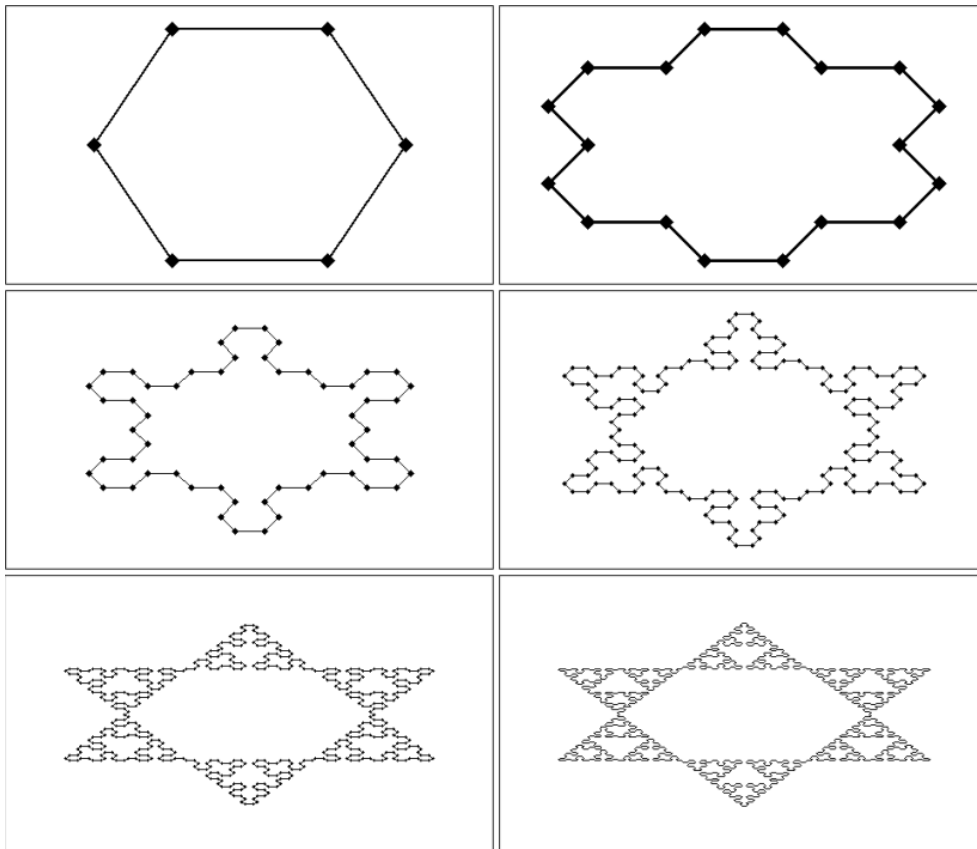


Figure 2.4: David Tour Exact Solutions - Orders 0 to 5

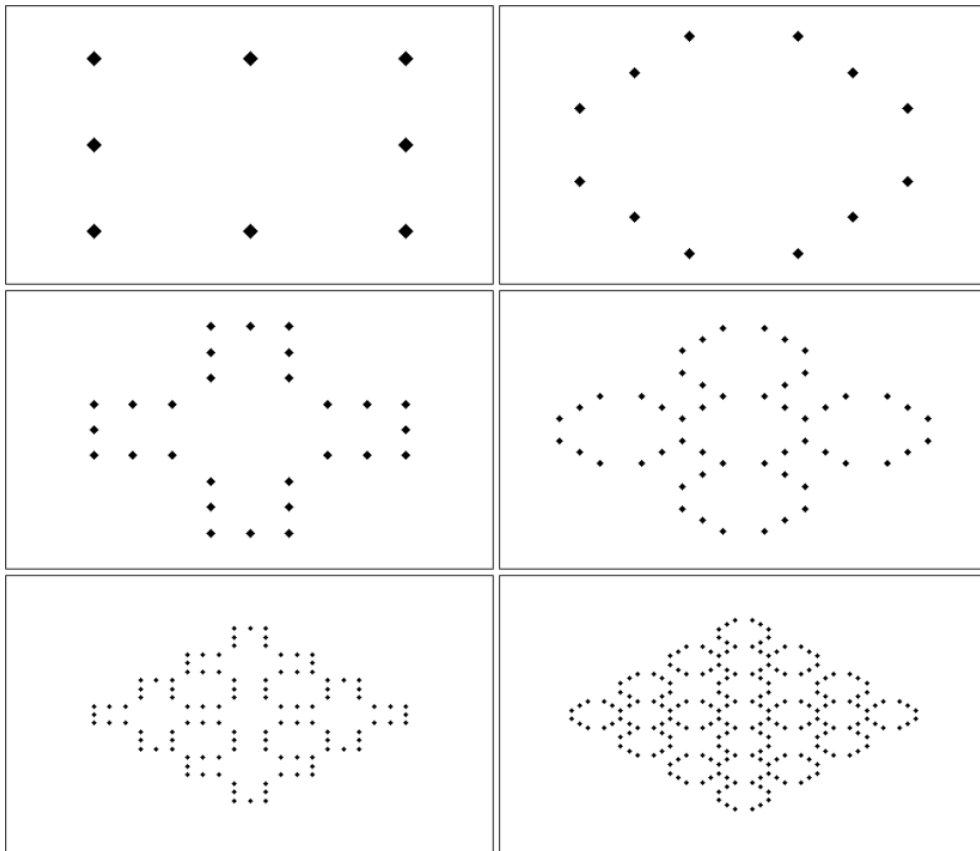


Figure 2.5: MPeano Coordinates - Orders 0 to 5

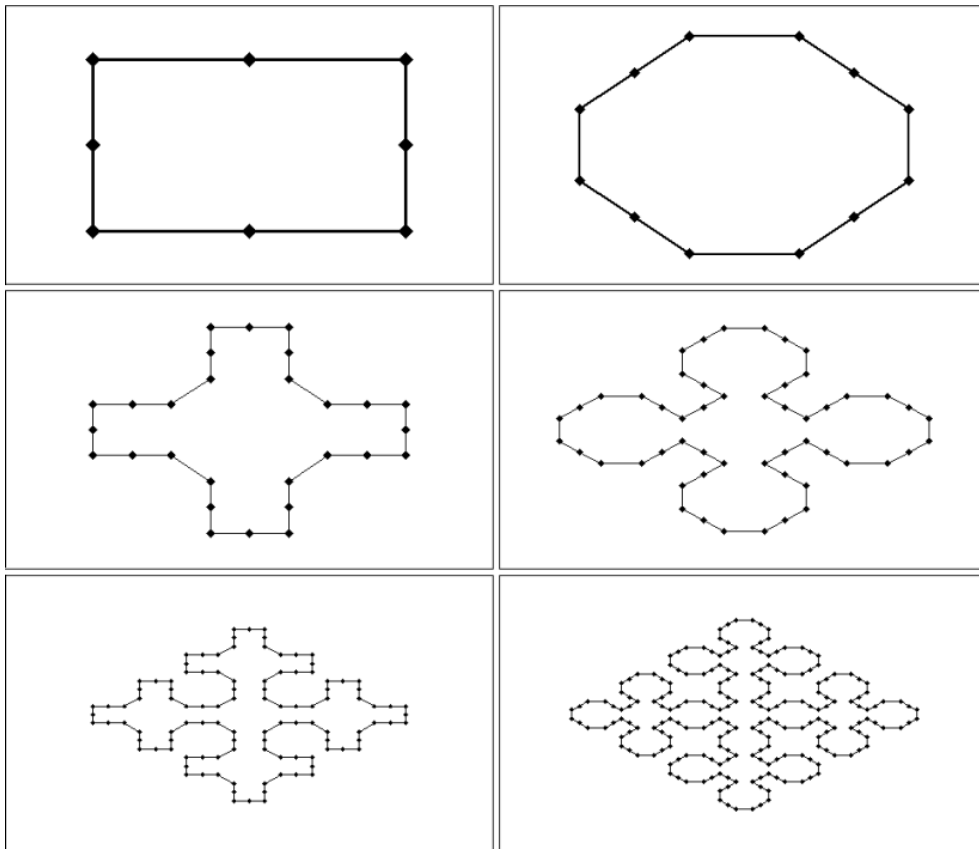


Figure 2.6: MPeano Exact Solutions - Orders 0 to 5

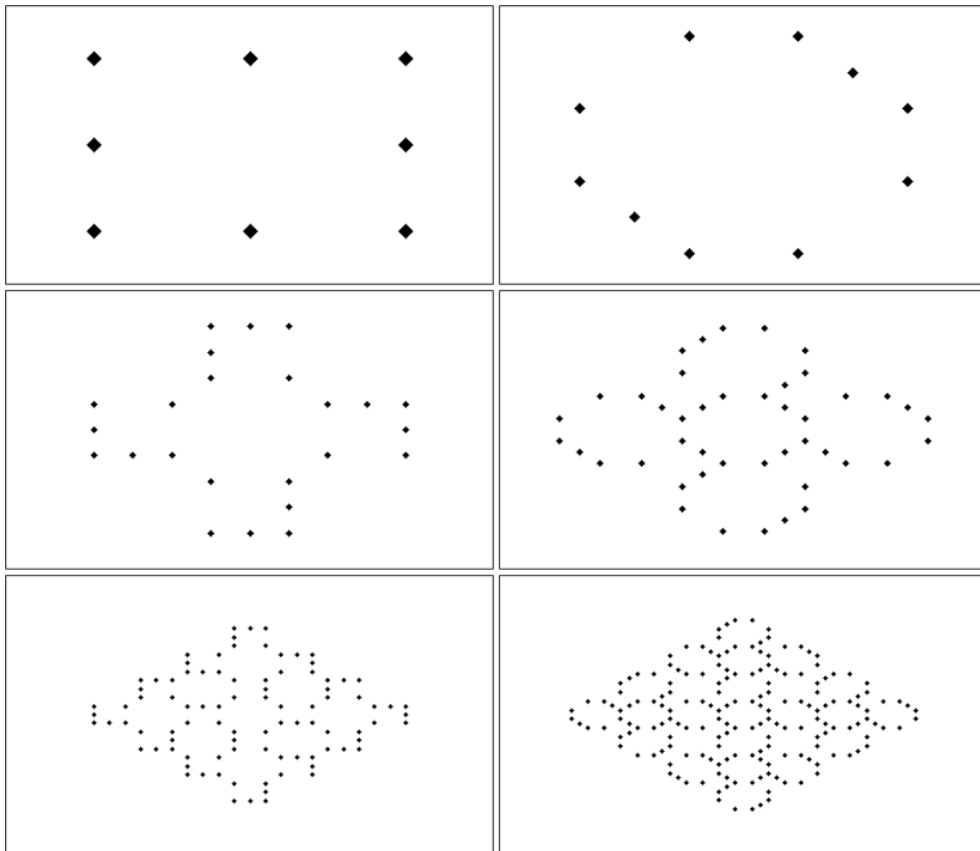


Figure 2.7: MN-Peano Coordinates - Orders 0 to 5

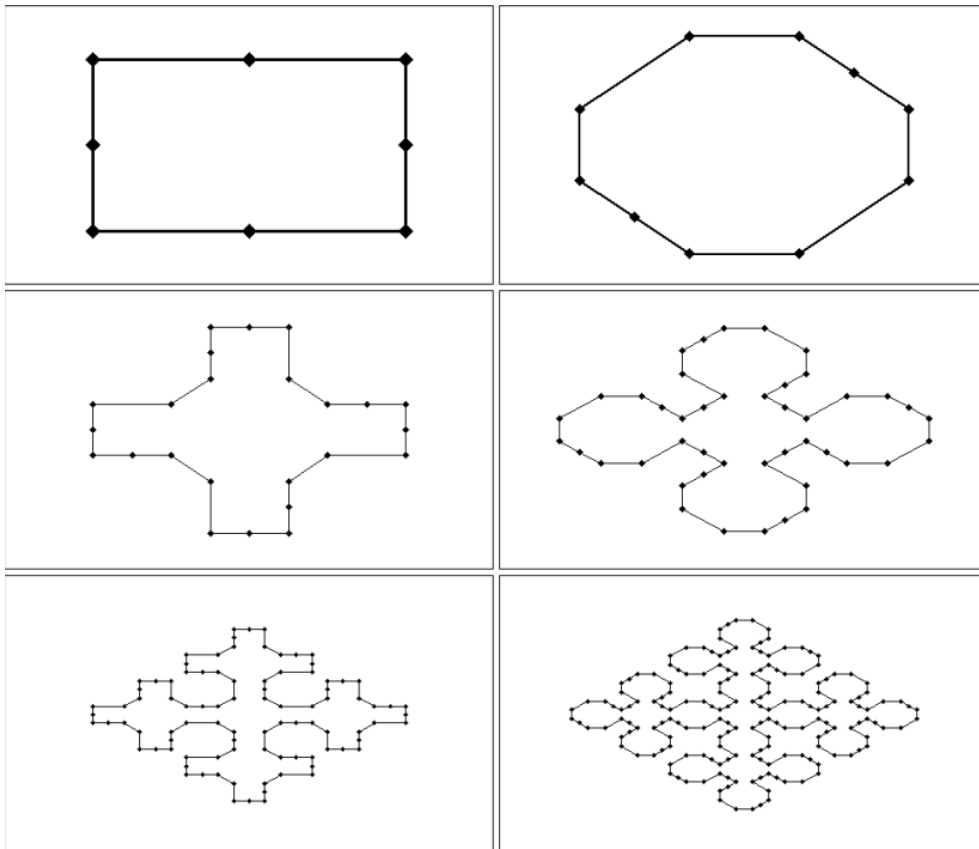


Figure 2.8: MN-Peano Exact Solutions - Orders 0 to 5

Chapter 3

Learning Classifier Systems

3.1 Chapter Overview

In this chapter the basics of the Learning Classifier Systems are explored, with a description of the most commonly used implementations. Then a more detailed definition is given for the system chosen for our experiments. This chapter concludes with a listing of data mining terminology and motivations that are to be known before the novel ideas in this thesis can be detailed.

3.2 Definition

Since Learning Classifier System (LCS) are a means to solving a **Classification Problem**, we begin by exploring its definition.

Definition 5. *A **Classification Problem** implies the need to separate a group of objects into differentiated classes, based on defining qualities that set them apart, and that make up the essence of each group. Situations ranging from a child's game of recognizing object patterns (squares, circles) to cellular analysis (when discerning between damaged and healthy cells) can be regarded as classification problems. Many mathematicians and computer scientists try to analyze situations that fit this criteria to model the problem into a clear set of functions and algorithms, to allow for a better chance of finding solutions that are more accurate and easier to obtain.*

Learning Classifier Systems were first mentioned around 1978, being [13] the most cited source of its definition. Today they are established as a good option when trying to solve a wide range of problems ([12],[31],[2],[3],[30]). First of all, an LCS is a machine learning technique. This means that the system can, if

presented with a task, increase its initial performance at the task as it gains experience. Considerations that arise here are:

1. The task has to be measurable, so that the system's efficiency can be evaluated
2. Experience has to be defined, and be measurable as well. This can imply registering time, number of attempts at the task, space of inputs already known to the system, etc.
3. The environment associated with performing the task needs to be modeled, so that environmental information can be used as input by the system.

In the particular case of LCS, the task is to classify a set of input data, normally defined as an array of properties (for **example** classify animals by means of their height and weight).

Since they were presented, many systems have been designed that fall into this category. The learning procedure is traditionally achieved by considering rules of the form **IF condition THEN action**, where the condition is evaluated against an instance of the input data, and the action implies classifying the current input with the value stated in the rule.

In the case of the animal classification **example** stated above, we could have rules like (*IF height < 1m and weight = 3kg THEN cat*) and (*IF height < 0.5m and weight > 4kg THEN iguana*).

The trick while developing these systems is to find an efficient way of evolving rules like these and reach a set of rules that together classify with high accuracy the whole input space (environment).

What remains to be said about them is that since its beginnings these systems have taken advantage of many techniques from the artificial intelligence area. The specific case of LCS we have chosen follows the *Pittsburgh approach*, which means a **Genetic Algorithm** (GA) is used as a heuristic to guide the classification process.

Genetic Algorithms are an idea central to **Evolutionary Computation** (EC). The field of EC brings ideas from the biological sciences such as *DNA coding*, *survival of the fittest* and *trait inheritance*. These ideas can be employed by themselves to solve complex problems ([7],[8],[6],[16],[17],[18],[29],[11]). A genetic algorithm is composed of the following elements:

1. An initial population of individuals, with each representing a possible solution to the problem
2. Generations: the iteration step that identifies one group of individuals from the next (offspring), usually fitter than the parents. In a GA, the "result" is the last generation spawned when the stopping criteria is reached.

3. A criteria for generating offspring from one or more individuals to give rise to the next generation.
4. A criteria for generating small mutations, so as to ensure diversity from one generation of individuals into the next.
5. A fitness criteria, that will ideally be higher in average among individuals from one generation with regards to the previous one.
6. A stopping criteria

These ideas are applied in the case of LCS to the rules described before. Next is a detail of the definition of GA parameters for an LCS:

1. The population to evolve is a collection of rule sets as those defined before. It is common to represent the rulesets as binary strings, with different sections representing different rules. For **example** for a rule predicating over 2 properties A and B with a possible value between 0 and 3, 2 bits can be used for each, meaning that 1100 is a ruleset expressing " $A = 3 \& B = 0$ ".
2. A generation is composed of the current collection of rule sets. It stands that every rule set has at least one rule matching each possible class, so that each individual (rule set) can always classify (even if incorrectly) all of the input data. Some optimizations (described later) ignore one of the classes and consider it a Default Class, allowing for smaller rule sets.
3. In the rules used as example above, the offspring criteria could be as simple as taking some rules from two different rule sets and join them to create a new complete rule set, or as complicated as creating new rules with the conditions from one rule set and the actions from another, mixing them with a probability function. Special care must be taken when mixing this rules, since the result must be a valid ruleset.
4. The mutation procedure usually implies a slight change in numerical values, or, if the rules are represented by a byte string, some 0's are changed into 1's or vice versa. As above, the validity of the resulting ruleset must be checked.
5. A fitness formula indicates the ability of the rule set to classify the input data correctly, helping to discard those rule sets that classify incorrectly or are unnecessarily more complex than others already present, allowing the system to evolve more compact and accurate rule sets.
6. Learning Classifier Systems stopping criteria depends on the implementation considered. The model used for this thesis takes a parameter indicating the number of iterations to run, but others could well stop when a certain

classification accuracy is reached, or the rule sets presents a certain property.

Aside from the EC component, Learning Classifier Systems require Machine Learning ideas to evaluate and correct the accuracy of the rules. The studied case employs Reinforcement Learning, which means that the input data already contains the expected classification (requiring the data to be supervised when gathered) and the system tries to predict the class based on the rules, to check then against the supplied classification, adjusting its internal values accordingly (depending on whether it was right or not). The specifics of such procedures vary greatly from one LCS to another, and so are discussed more in depth when analyzing the particular system used in this work.

3.3 Variations

Although the previous sections detailed the basics of LCS, they only hint the wide range of variations implemented as of today. Since this thesis is based on the results of a Pittsburgh LCS, we note next other state of the art approaches before delving deeper in the specifics of our chosen system. For a more detailed introduction to these variations, please refer to [4, 1].

3.3.1 Michigan approach

The Michigan Approach was developed before 1990. Around that time, the implementation known as Zeroth Level Classifier System (ZCS) was the most representative. Nowadays the most commonly seen LCS is its successor XCS, a newer model that uses a different approach to fitness computation, presenting a more robust framework that improves overall accuracy.

The most important difference between the Michigan and the Pittsburgh approach is the part rules play in the evolution of the system. While the Pittsburgh approach considers rule sets as individuals in a GA context, as was described above, the Michigan approach considers rules as individuals, which constantly try to classify instances presented to the system. This difference can be seen a change in the philosophy behind the program structure. In the Pittsburgh approach, rule sets battle against each other to attain the capacity to represent the whole of the input space, while being also as compact as possible. The Genetic Algorithm employed takes care of the evolutionary process. On the other hand, the Michigan approach implies a coordination between the rules to emerge with a complete rule set, and here the Genetic Algorithm plays a lesser role.

Wilson's implementation of ZCS [33] implies the constant evaluation of such classifications, where the different rules are awarded "points" when they correctly classify an instance. Similar rules are grouped and their points distributed

equally. Points are also lost when a rule is not used. At certain intervals, a GA is used to create offspring from the most promising rules, which replace those with lower classification scores. It can be seen from this that the role of the GA in the system is quite different from the highlight it occupies in the Pittsburgh approach.

XCS improves over ZCS in the fitness computation of the rules. Apart from predicting a class, each individual (rule) has a number of attributes associated to it that take part in each cycle of the system. The system also includes some global values regarding learning rates, deletion thresholds and the activation of the GA.

One interesting aspect of this approach is that when the set of rules is presented with an instance not covered by any of the rules, a process called *covering* takes place, creating new rules to take the instance in consideration. As many rules as needed are added, so as to ensure that every class has at least one rule vouching for it, though obviously other factors such as the fitness of the rules will determine the final prediction.

3.3.2 Iterative Rule Learning approach

The Iterative Rule Learning approach implies a number of runs of a GA, where each run ensures that a part of the input space will be covered by newly generated rules, so that after N runs a complete rule set is obtained.

An important issue with this approach is that since each run will provide one rule, measures have to be taken to avoid over specification. Also, the system has to provide a consistent balance between the coverage increased at each iteration and the accuracy of the rules produced.

3.4 GAssist

The evaluation of the proposed models will be tested by using **GAssist**, a Pittsburgh like learning classifier system, introduced in [1]. Although GAssist contains a full fledged LCS, most of its basic architecture is based on a simpler (and earlier) system known as GABIL. Since both implementations share a common core of functionality and behavior, we will first detail the basics of GABIL, and then proceed to detail the optimizations and further classification considerations that GAssist properly introduces.

3.4.1 GABIL

GABIL is described in detail in [14]. The interested reader is referenced to this paper for its implementation. However, we will mention the features of GABIL that are used in GAssist:

1. Fitness function (squared accuracy): the fitness of an individual (RuleSet) is computed as $(\frac{\#instances\ correctly\ classified}{total\ \#instances})^2$.
2. Nominal Dataset representation: in the case of classification based on attribute A1 with 4 possible values (X1,X2,X3,X4) and A2 with 2 possible values (Y1,Y2), the rule represented by the binary string 1101 01 implies the predicate “(A1 is X1 or X2 or X4) and (A2 is Y2)”.
3. Semantically correct crossover operator: different cut points can be chosen in the parent rules, as long as they are placed in the same position inside the offspring rule.
4. Bit flipping mutation for the nominal representation

3.4.2 Additional Framework Considerations

Apart from the details mentioned previously, GAssist presents the following features that are to be taken into consideration:

1. Instances will be classified according to the first rule that make them true.
2. Missing values in datasets are substituted considering instances belonging to the same class, with the most frequent value for nominal attributes, and the average value for real valued attributes.

3.4.3 Contributions

GAssist improves the state of the art for Pittsburgh Learning Classifier Systems by integrating into GABIL the following features:

1. Explicit and static default rule: the best default rule in terms of accuracy and reduction of search space is selected and transparently used throughout the system.
2. Adaptive discretization intervals (ADI): this technique implies an evolving discretization of real-valued attributes, that can vary for each rule and attribute, and which can also split and merge as the system does its job.
3. Incremental learning by alternating strata (ILAS): a windowing technique for generalization and run-time reduction.
4. Bloat control and generalization pressure methods: GAssist presents a rule deletion operator, and a fitness function that considers the content of the individuals in guiding the exploration process.

3.5 Data Mining

As a fundamental part of this work intends to gain a better understanding of the qualities that help define TSPs in terms of their likeness to other well known graph structures, we will review and remember from time to time the data mining concept of **features**.

A **feature** is a measurable property of an object. While described in that general way it seems to encompass almost anything, the objective when performing Data Mining tasks is to identify the features of the subject of study that allow it to be distinguished from other objects that are to be treated differently. Features can be represented by numbers (real or integer valued), unrestricted string (i.e. names) or discrete partitions of a set, to name a few possibilities.

The choice of the right features for a Data Mining / Pattern Recognition / Classification problem is essential to the success of the solution proposed.

Chapter 4

LCS for classifying TSP optimal edges

4.1 Chapter overview

This chapter begins the novel proposals which provide the main objective for this thesis. To put it shortly, we will look into a different way of solving instances of TSP.

In this and the following chapters, several **models** are revised, which *map the coordinates that make up an instance of the FTSPs into attributes that can be used as input for GAssist*. Their objective is to help in determining, from a group of already existing algorithms, which to use for a given problem.

Each model presented will use specifically generated instances of FTSPs, and will identify properties from the collection of coordinates, which will then be used to provide answers that can help us find the solution to the instance. This, of course, is only the beginning, since from the results of this work, we intend to help create the knowledge to solve *any* instance of TSP, and not only the sample Fractal ones presented here. This generalization will take place in the following chapters.

The remainder of the present chapter is structured as follows. First the simple Triplets model will be presented, as a way of helping the reader grasp the possibilities of using GAssist as an indicator of how to solve the Traveling Salesman Problem. After presenting its structure, the experimentation proposed will be explained, followed by the results of such experiments. Then some research ideas will be listed, as future work derived from the model is possible.

4.2 Triplets

4.2.1 Description

The first model proposed is the one called **Triplets**. Its creation and structure are kept simple, since this representation allows for the beginning of our research into the feasibility of applying learning classifier systems to a combinatorial optimization problem such as TSP.

The objective of this model is for GAssist to learn, from a group of edges, which are to be part of a solution to a TSP and which are to be excluded from it. Since by creating one of the FTSPs presented we know the edges in the exact solution, we can easily create a training set mixing these edges with random edges not included in the solution, thus obtaining an important group of samples to train a system such as GAssist.

As was mentioned while describing GAssist in 3.4, any model proposed would need to be explained in terms of GAssist instances, with each having a number of attributes with their corresponding values and a classification.

In the case of 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: $(1, 1) \rightarrow (1, 2) \rightarrow (2, 1) \rightarrow (1, 1)$. This means that the GAssist attributes described would have the following values (Table 4.1)

Instance #	Edge 1	Edge 2
1	$(1, 1) \rightarrow (1, 2)$	$(1, 2) \rightarrow (2, 1)$
2	$(1, 2) \rightarrow (2, 1)$	$(2, 1) \rightarrow (1, 1)$

Table 4.1: Attributes for an instance of the Triples model

Also, the tour can be seen backwards, which means that the attributes presented in Table 4.2 are valid paths of size 2 of an optimal solution as well.

Instance #	Edge 1	Edge 2
3	$(1, 1) \rightarrow (2, 1)$	$(2, 1) \rightarrow (1, 2)$
4	$(2, 1) \rightarrow (1, 2)$	$(1, 2) \rightarrow (1, 1)$

Table 4.2: Attributes for an instance of the Triples model - Reverse

To complete the possible attributes of the model, we must consider what happens when the initial vertex changes, leading us to add the cases listed in Table 4.3.

Instance #	Edge 1	Edge 2
5	$(2, 1) \rightarrow (1, 1)$	$(1, 1) \rightarrow (1, 2)$
6	$(1, 2) \rightarrow (1, 1)$	$(1, 1) \rightarrow (2, 1)$

Table 4.3: Attributes for an instance of the Triples model - Permutations

These six attribute sets complete all valid permutations of paths of size 2 in the tour $(1, 1) \rightarrow (1, 2) \rightarrow (2, 1) \rightarrow (1, 1)$ used as example.

This of course means that redundancy will exist between instances, but as will be seen 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).

As an example of the proposed experiment, we will consider the first order of Koch Tour. By using a coordinates generator, we obtain the edges presented in Table 4.4. These edges are then presented in Figure 4.1, which confirm that such edges form a Koch Tour of first order (for more details about its construction, refer to section 2.5.3).

Instance #	Edge 1	Edge 2
1	$(-15, 9) \rightarrow (-10, 0)$	$(-10, 0) \rightarrow (-15, -9)$
2	$(-10, 0) \rightarrow (-15, -9)$	$(-15, -9) \rightarrow (-5, -9)$
3	$(-15, -9) \rightarrow (-5, -9)$	$(-5, -9) \rightarrow (0, -18)$
4	$(-5, -9) \rightarrow (0, -18)$	$(0, -18) \rightarrow (5, -9)$
5	$(0, -18) \rightarrow (5, -9)$	$(5, -9) \rightarrow (15, -9)$
6	$(5, -9) \rightarrow (15, -9)$	$(15, -9) \rightarrow (10, 0)$
7	$(15, -9) \rightarrow (10, 0)$	$(10, 0) \rightarrow (15, 9)$
8	$(10, 0) \rightarrow (15, 9)$	$(15, 9) \rightarrow (5, 9)$
9	$(15, 9) \rightarrow (5, 9)$	$(5, 9) \rightarrow (0, 18)$
10	$(5, 9) \rightarrow (0, 18)$	$(0, 18) \rightarrow (-5, 9)$
11	$(0, 18) \rightarrow (-5, 9)$	$(-5, 9) \rightarrow (-15, 9)$
12	$(-5, 9) \rightarrow (-15, 9)$	$(-15, 9) \rightarrow (-10, 0)$

Table 4.4: Triplets model Attributes - Koch Tour - Order 1

To transform the vertices into GAssist instances, we apply the following procedure. Taking the first pair of edges in Table 4.4 $(-15, 9) \rightarrow (-10, 0)$ and $(-10, 0) \rightarrow (-15, -9)$, we observe that 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: $(-15, 9), (-10, 0), (-15, -9)$. The NUMERIC type of GAssist is used for

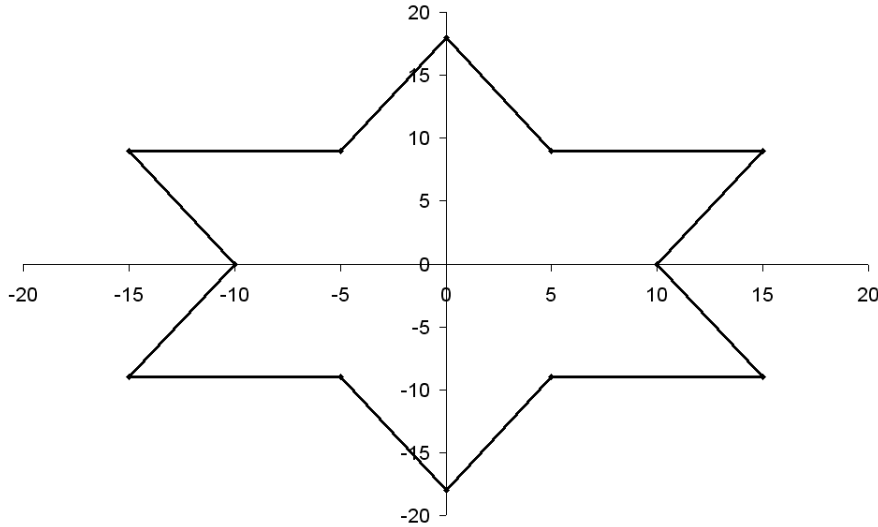


Figure 4.1: Koch Tour - Order 1

six attributes: **Previous X**, **Previous Y**, **Current X**, **Current Y**, **Next X** and **Next Y**. The example gives us the values: $-15, 9, -10, 0, -15, -9$ respectively. The class for this instance is “Included”. Repeating this procedure for all the 12 vertices, Table 4.5 is built.

#	Prev. X	Prev. Y	Cur. X	Cur. Y	Next X	Next Y	Class
1	-15	9	-10	0	-15	-9	Included
2	-10	0	-15	-9	-5	-9	Included
3	-15	-9	-5	-9	0	-18	Included
4	-5	-9	0	-18	5	-9	Included
5	0	-18	5	-9	15	-9	Included
6	5	-9	15	-9	10	0	Included
7	15	-9	10	0	15	9	Included
8	10	0	15	9	5	9	Included
9	15	9	5	9	0	18	Included
10	5	9	0	18	-5	9	Included
11	0	18	-5	9	-15	9	Included
12	-5	9	-15	9	-10	0	Included

Table 4.5: Triplets model Instances - Koch Tour - Order 1

Recalling the number of permutations presented in the example of the model, 12 instances are missing, which belong to the reverse tour. These are also to be considered and can be seen in Table 4.6.

Since the GAssist instances presented are by definition all of the instances belonging to class “Included”, we can easily generate instances for the “Not in-

#	Prev. X	Prev. Y	Cur. X	Cur. Y	Next X	Next Y	Class
13	-15	-9	-10	0	-15	9	Included
14	-5	-9	-15	-9	-10	0	Included
15	0	-18	-5	-9	-15	-9	Included
16	5	-9	0	-18	-5	-9	Included
17	15	-9	5	-9	0	-18	Included
18	10	0	15	-9	5	-9	Included
19	15	9	10	0	15	-9	Included
20	5	9	15	9	10	0	Included
21	0	18	5	9	15	9	Included
22	-5	9	0	18	5	9	Included
23	-15	9	-5	9	0	18	Included
24	-10	0	-15	9	-5	9	Included

Table 4.6: Triplets model Instances - Koch Tour - Order 1 - Reverse

cluded” class. We do so by randomly choosing vertices and combining them so as to generate different instances. Twenty-four such instances are presented in table 4.7. Since the Koch Tour coordinates were generated with a method that ensured an 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.

Definition 6. ***K-fold cross validation** is an experimentation method used for the testing of algorithms. It consists of dividing the sample inputs in \mathbf{K} independent sets and then consider in turn each set as the testing data, while the remaining $\mathbf{K} - 1$ are used as training data. After \mathbf{K} such repetitions, an average for the accuracy of the algorithm can be expressed.*

In order to evaluate the model, we focus now on building groups of instances to perform a cross validation experiment. Table 4.8 shows a partition of the 24 instances presented so far into 4 groups.

This ensures that every group has 6 instances with class “Included” and 6 with class “Not included”. With these groups we perform a 4-fold cross validation. Each group was used as testing set while the rest were used as training set, and the experiment was repeated 10 times.

#	Prev. X	Prev. Y	Cur. X	Cur. Y	Next X	Next Y	Class
25	5	9	10	0	0	-18	Not inc.
26	-5	9	5	-9	10	0	Not inc.
27	15	-9	-5	9	-10	0	Not inc.
28	-5	-9	10	0	0	18	Not inc.
29	5	9	-10	0	-15	-9	Not inc.
30	15	9	5	9	-5	-9	Not inc.
31	10	0	0	18	-10	0	Not inc.
32	0	-18	5	-9	5	9	Not inc.
33	-15	-9	5	-9	-15	9	Not inc.
34	-15	9	0	18	-10	0	Not inc.
35	-15	-9	5	-9	-10	0	Not inc.
36	15	-9	0	18	-15	-9	Not inc.
37	15	9	15	-9	-5	9	Not inc.
38	0	-18	-15	9	-10	0	Not inc.
39	0	-18	-5	-9	15	-9	Not inc.
40	-10	0	15	9	-15	9	Not inc.
41	5	-9	-15	9	10	0	Not inc.
42	-5	-9	0	18	-15	9	Not inc.
43	-15	9	0	-18	-5	9	Not inc.
44	-15	-9	5	9	15	9	Not inc.
45	5	-9	-10	0	10	0	Not inc.
46	5	-9	-15	9	-5	-9	Not inc.
47	0	-18	15	-9	-15	9	Not inc.
48	0	18	5	9	5	-9	Not inc.

Table 4.7: Randomly generated instances not present in Koch Tour order 1

Group	Instances included
1	14,2,22,10,18,6,25,26,27,28,29,30
2	13,1,21,9,17,5,31,32,33,34,35,36
3	24,12,20,8,16,4,37,38,39,40,41,42
4	23,11,19,7,15,3,43,44,45,46,47,48

Table 4.8: Triplets model Groups - Koch Tour - Order 1

4.2.2 Experimental Results

Table 4.9 shows the results obtained for the experiment detailed in the previous section. Each value presented details the average of the testing accuracy for the 10 runs for each fold built.

The same idea was then repeated for all four curves for orders 1 to 6, with results shown in Figure 4.2. Each point is computed with a value equivalent to the final averaged value of the previous table. This means that the testing accuracy averaged over 10 runs of GAssist for a fold (which belong to a particular order

Group	Testing Accuracy (%)
1	54.16
2	38.33
3	55.83
4	56.66
Average	51.25

Table 4.9: Triplets model Testing Accuracy Results - Koch Tour - Order 1

of a particular curve), is then averaged for all 4 folds, giving a value that is then used to build the mentioned figure. 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 chapters 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, impeding 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 instance used can determine the class of another. This is 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.

4.2.3 Learning space

We are interested now, after analyzing the results of the experiments proposed, to better understand the meaning of the accuracies obtained.

Given a TSP instance with n coordinates, the complete graph $K(n)$ for it would contain $\frac{n(n-1)}{2}$ different edges. Of this edges, only n are part of the optimal tour. This means that in a positive cases over total cases we have: $\frac{n}{\frac{n(n-1)}{2}} = \frac{2n}{n(n-1)} = \frac{2}{n-1}$. Giving a $\frac{2}{n-1}$ chance of finding an edge included in the optimal tour when picking randomly from all of the edges of a complete graph. This is obviously only of interest for $n \geq 3$ since for lower values there is no possible solution.

This implies that if a program were to decide blindly how to form a solution to the TSP, an initial probability would be $(\frac{2}{n-1})^n$ if it didn't even have the

intelligence to avoid repetitions, and it would drop to $\frac{(\frac{n(n-1)}{2})!}{n!((\frac{n(n-1)}{2})-n)!}$ for the case of a combination without repetitions allowed. The previous formula does not just seem shocking, it also implies, for the first order of MPeano, a probability of 1 in 4922879481520 to find the optimum path. Once a program is developed with the ability to allow only valid Hamiltonian cycles, the choice is reduced even more, to only $\frac{(n-1)!}{2}$. In the example of MPeano order 1, this means 1 in 19958400.

These values must be kept in mind when considering the complexity of the problem, and the importance of using the results of this work to approach as best as possible to the optimum path. Chapter 7 provides a means to apply the theoretical ideas developed in order to reach out for this ambitious goal.

4.2.4 Conclusion

Some conclusions can be obtained from the results. It can be guessed that since the number of coordinates in the curves increases exponentially with each order, GAssist has more information from where to create the rules that are then used for testing purposes.

There remains, however, interest in understanding how the lack of data impacts the rules. Figure 4.3 shows some of the rules obtained by GAssist, where a failing of the model can be observed: the absolute positioning of the coordinates is used directly to decide. The reader can observe that the values presented in the rules are distant to the coordinate values presented in the examples. This repositioning, caused by using different starting drawing points in the experiments than in the examples, confirms that applying the rules in a different scenario will cancel all chances of succeeding at the classifying task in a generalized manner, as is the objective in this and any other classification problem.

4.2.5 Future Work

With a solid efficiency in training, this model can be used to deduce how to “fill in” a curve with missing connections. Some modifications can be done to it as well:

- Normalization of Coordinates
- Expression of triplets as relative distance from a Coordinate to its previous and next one, thus reducing the size of the instance and improving the results achieved.
- Improvement of the “Not Included” Instances, obtained from Hamiltonian Tours instead of random edges.

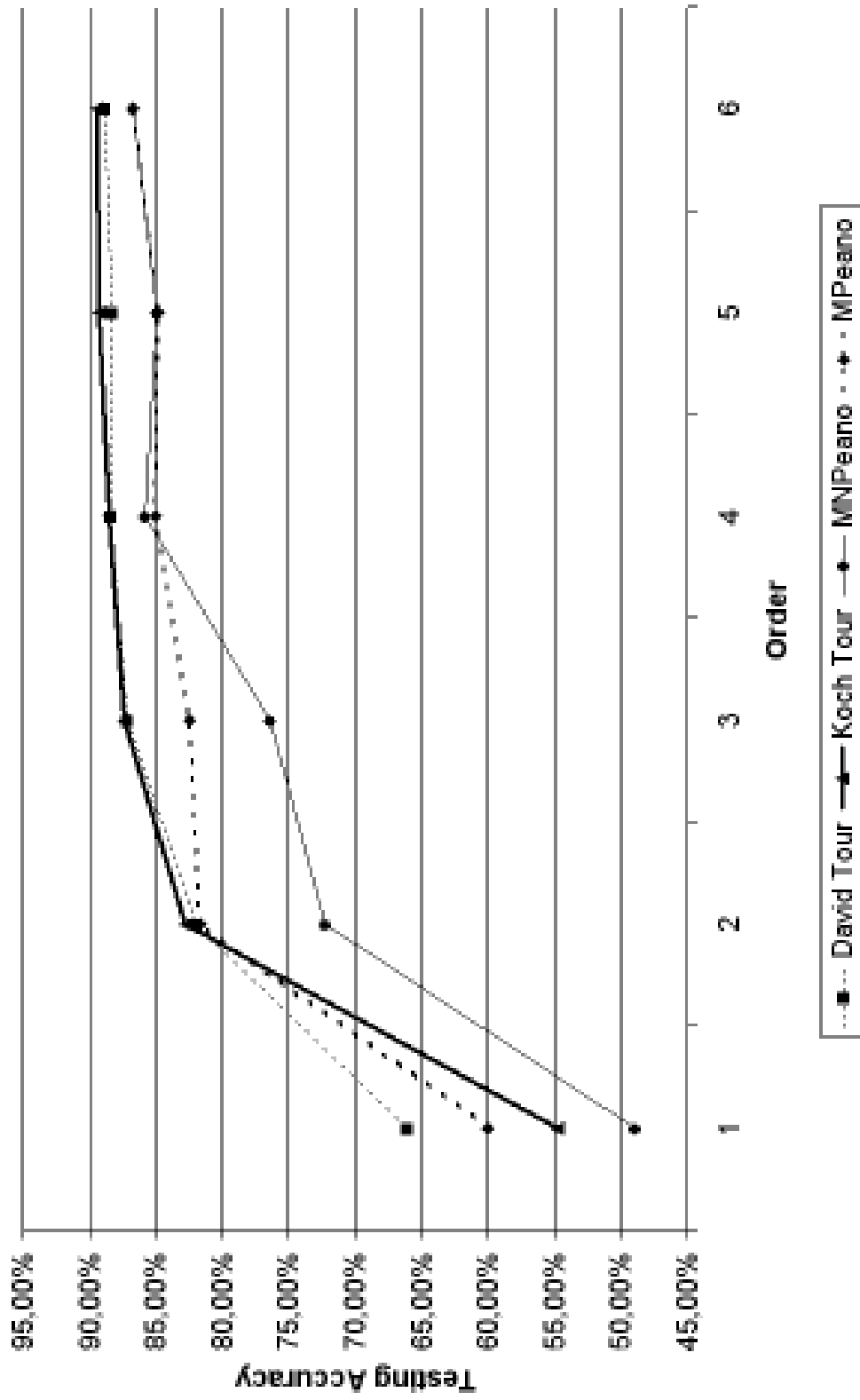


Figure 4.2: Triplets Model - Testing Accuracy

```
0:Att previousX is [>520.7142857142856] |
  Att nextX is [<520.7142857142856] |
  Not Included
1:Att currentX is [<-729.0] |
  Att nextX is [>-729.0] |
  Not Included
2:Att previousX is [<0.0] |
  Att currentX is [>0.0] |
  Not Included
3:Att previousY is [<1458.0] |
  Att currentY is [>1640.25] |
  Not Included
4:Att currentY is [<-1458.0] |
  Att nextY is [>-1345.846153846153] |
  Not Included
5:Default rule -> Included
```

Figure 4.3: GAssist rules for the 5th run of Koch Tour Order 6 on group 1

Chapter 5

TSP features for instance classification

5.1 Chapter overview

Here we continue with the search for a model that can grasp interesting qualities in a TSP, so that they can be used to classify the different classes. This chapter details the Global Statistics model, which includes the most important proposals of this work, since is from the structure and attributes here shown that all of the next models will be based. As with the Triplets model, we present first a detail of its implementation, then suggested experiments and results obtained, to conclude finally with ideas yet to be considered.

5.2 Global Statistics

5.2.1 Description

With the understanding of how GAssist can help our research, and the objective of finding a model that describes graphs without resorting to absolute references to its coordinates, we will explore a new model, to which we will refer to as **Global Statistics**. This model aims at describing curves by means of their features, that is, values obtained from the analysis of the graphs, that summarize them and allow for a comparison between them. Our final goal is to, for example, consider all of the coordinates of a second order Koch Tour, and deduce from them that they describe a Koch Tour graph. This is of course easy from our point of view, since we already know where those coordinates came from and so the question is trivial. But, if from that information, GAssist could deduce rules that associate the coordinates with the class of curve, we could know, given some

coordinates from an unknown source, if they described a Koch Tour. This means that we would know immediately how to present the exact solution to the graph, since as proved in [22], there is a known algorithm to solve the TSP when the graph is known to describe a Koch Tour.

The difficulty lies, of course, in deciding how to present those coordinates to GAssist in a way that those “features”, as they are referred to in data mining, can be compared between classes, between different orders of the same class, and also be compatible with any type of graph, so that our model can then be expanded to face more challenging tasks.

Taking into consideration the ideas above, we present now a way to express our classes (MPeano, MNPeano, David Tour, Koch Tour), in terms of GAssist attributes, so as to ensure a good classification. Note that we want to identify a curve by the value of its attributes, despite which order it represents. This means that we are to understand and take advantage of, for example, the similarities between the first and third orders of Koch Tour, and their differences in comparison with the fifth order of MPeano. As we are considering 4 classes, and each class can be computed within reasonable time only up to its sixth order, a total of just 24 GAssist instances are available for this model. Improved models in forthcoming sections will address this issue.

Attributes Proposed

We propose the following GAssist NUMERIC attributes:

- **Number of Cities**
- **Maximum X Component**
- **Maximum Y Component**
- **Average Spread from X Axis Center**
- **Average Spread from Y Axis Center**
- **Average of Nearest Neighbors**

Number of Cities

This is a straightforward value obtained by counting the amount of coordinates in the curve.

Maximum X/Y Component

Since this model aims at avoiding 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 \mathbf{x} and \mathbf{y} value are 0, but the highest values need not be exactly 1 (ie. when the figure described by the curve is not symmetrical from its center). These values are computed as follows:

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

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

Average Spread from X/Y Axis Center

These values intend to express a general idea 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 following formula details the idea:

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

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

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

As a practical example we will consider a curve with the following coordinates: (1,1),(2,3),(5,4). Table 5.1 presents the maximum, minimum and center values:

X_{min}	X_{max}	Y_{min}	Y_{max}	$center_x$	$center_y$
1	5	1	4	3	2.5

Table 5.1: Global Statistics Centered Coordinates Example

This means that our coordinates have the following spread values (Table 5.2):

Coordinate	$Spread_x$	$Spread_y$
(1,1)	0	0
(2,3)	0.5	0.66
(5,4)	0	0
Average	0.16	0.22

Table 5.2: Global Statistics Spread Example

Average of Nearest Neighbors

This 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.

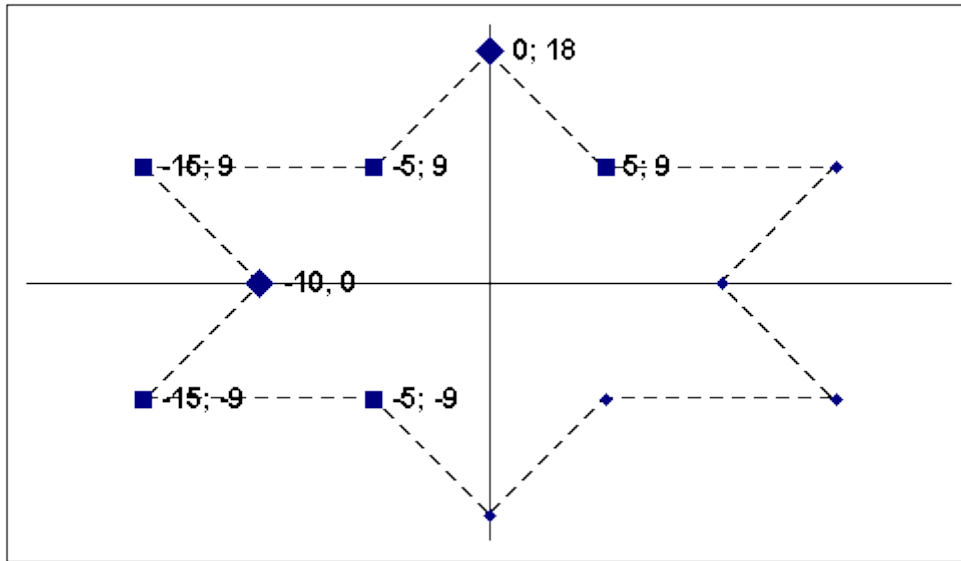


Figure 5.1: Koch Tour Order 1 Nearest Neighbors

Figure 5.1 shows the first order of Koch Tour with some highlighted cities. Note that the city at $(-10,0)$ has 4 other cities $((-15,9), (-5,9), (-15,-9)$ and $(-5,-9))$ at the same distance, all of them nearer than the rest. This means that $(-10,0)$ has 4 nearest neighbors. In the case of the city at $(0,18)$, the number of nearest neighbors is 2: $(-5,9)$ and $(5,9)$.

Instance Example

In order to summarize the previous definitions, the transformation of a FTSP into a GAssist instance is reviewed next.

We will consider the Koch Tour used in this section, since some of its nearest neighbours have already been reviewed.

Number of Cities is 12 since this is the number of points in the graph. In Figure 4.1, as well as in 5.1, it can be seen that the difference between the maximum and minimum X component is 30, while the difference in Y is 36. This means that after normalization of the coordinates in a 1×1 square, the **Maximum X Component** is 0,8333 (that is, $\frac{30}{36}$), while the **Maximum Y Component** is 1.

Given the even distribution of the points in this and most FTSP, it is clear that the spread values should be near 0.5. In this case, **Average Spread from X Axis Center** is 0.4444 and **Average Spread from Y Axis Center** is 0.5. The **Average of Nearest Neighbors** was computed as explained previously, giving a final value of 1.8333.

Folding

Now that the attributes have been presented, we continue to detail a way to experiment with the model and discover its potential. First, as we intend to continue applying K-Fold Cross Validation, we need to group the available instances. As was previously explained, this means 24 curves have to be taken into consideration, with 6 belonging to each class. The solution proposed is to perform a “leave one out” cross validation, so that each fold contains one instance. Each of the 24 curves is transformed into a GAssist instance by computing the described attributes and indicating the class they belongs to.

5.2.2 Experimental Results

Before presenting the results, it must be remarked that since we are measuring testing accuracy over one instance in each run, the result of a run can only be a 0% or a 100% accuracy, that is, whether GAssist was correct or not at trying to deduce the class of an instance based on the knowledge of the other 23 instances built.

We present the average of 10 runs for each instance, grouped by class.

The results obtained for this model (Table 5.3) provide an excellent accuracy for Koch Tour 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 figure obtained. As regards MPeano and MNPeano, results show a much lower accuracy. This, however, does not come as unexpected, since both curves share almost all cities, and even a human observer can not do much to differentiate between them.

In order to prove that the cause of such low values during testing is the one proposed, the experiment was rerun twice, first without considering MPeano, and then without considering MNPeano, so as to avoid confusing GAssist.

Tables 5.4 and 5.5 confirm the hypothesis, while also maintaining the high accuracy for Koch Tour and David Tour.

Figure 5.2 shows the rules generated by GAssist during one run with all four classes included. It can be seen that the conditions for each classification are more complicated than the ones in figures 5.3 and 5.4, where either MPeano or MNPeano were absent from the data sets. These two figures show that in these cases the classification requires less conditions. Also, we can appreciate that the structure of the rules in both cases is almost identical, again confirming that the MPeano and MNPeano share the same shape even to the perception of the proposed model.

These results are indeed encouraging, as they provide proof that the attribute representation detailed is useful in classifying curves by their class with the use of GAssist.

Class	Order	Testing Accuracy (% Averaged over 10 runs)
MPeano	1	70
MPeano	2	50
MPeano	3	40
MPeano	4	70
MPeano	5	0
MPeano	6	20
MNPeano	1	80
MNPeano	2	0
MNPeano	3	60
MNPeano	4	0
MNPeano	5	50
MNPeano	6	90
Koch Tour	1	70
Koch Tour	2	100
Koch Tour	3	100
Koch Tour	4	100
Koch Tour	5	100
Koch Tour	6	100
David Tour	1	10
David Tour	2	40
David Tour	3	100
David Tour	4	90
David Tour	5	100
David Tour	6	100

Table 5.3: Global Statistics Testing Accuracy

5.2.3 Future Work

Although the next sections will be based on this model, some items are listed to be considered in other projects:

- Addition of new TSP classes: both similar and different to the existing ones. Classes with different structures would help determine the maximum amount of different classes this model can withstand before losing accuracy. On the other hand, the addition of classes similar to Koch Tour and David Tour could help understand if the confusion while classifying MPeano and MNPeano extends to other structures.
- Reduction of attributes: although the selected attributes have shown to be relevant while classifying the curves, some of them might be more important than others, and a reduction of attributes can help improve running times.
- MPeano differentiation from MNPeano: some additional attribute could be

Class	Order	Testing Accuracy (% Averaged over 10 runs)
MNPeano	1	100
MNPeano	2	60
MNPeano	3	100
MNPeano	4	100
MNPeano	5	100
MNPeano	6	100
Koch Tour	1	30
Koch Tour	2	100
Koch Tour	3	100
Koch Tour	4	100
Koch Tour	5	100
Koch Tour	6	100
David Tour	1	0
David Tour	2	80
David Tour	3	90
David Tour	4	90
David Tour	5	100
David Tour	6	100

Table 5.4: Global Statistics Without MPeano Testing Accuracy

devised to help break ties when GAssist has to decide which of these two curves an instance belongs to.

Class	Order	Testing Accuracy (% Averaged over 10 runs)
MPeano	1	90
MPeano	2	100
MPeano	3	90
MPeano	4	100
MPeano	5	100
MPeano	6	90
Koch Tour	1	40
Koch Tour	2	100
Koch Tour	3	100
Koch Tour	4	100
Koch Tour	5	100
Koch Tour	6	100
David Tour	1	10
David Tour	2	70
David Tour	3	100
David Tour	4	90
David Tour	5	100
David Tour	6	90

Table 5.5: Global Statistics Without MNPeano Testing Accuracy

```

0:Att maximumXcomponent is [<0.8645833333333333] |
Att xAxisCenterAveragedSpread is [<0.4448987538940812] |
KochTour
1:Att yAxisCenterAveragedSpread is [<0.6037949589351459] |
Att averageNearestNeighbours is [<1.904661148313493] |
MPeano
2:Att maximumXcomponent is [<0.9285714285714287] |
Att averageNearestNeighbours is [>1.9284551711309523] |
DavidTour
3:Att xAxisCenterAveragedSpread is [>0.51183800623053] |
MPeano
4:Default rule -> MNPeano

```

Figure 5.2: GAssist rules for the Global Statistics Model (With MPeano and MNPeano)

```
0:Att maximumXcomponent is [>0.979166666666667] |
MNPeano
1:Att maximumXcomponent is [>0.8484848484848485] |
DavidTour
2:Att xAxisCenterAveragedSpread is [>0.44453463203463217] |
DavidTour
3:Default rule -> KochTour
```

Figure 5.3: GAssist rules for the Global Statistics Model (Without MNPeano)

```
0:Att maximumXcomponent is [>0.9848484848484848] |
MPeano
1:Att maximumXcomponent is [>0.8571428571428572] |
DavidTour
2:Att xAxisCenterAveragedSpread is [>0.4523364485981311] |
DavidTour
3:Default rule -> KochTour
```

Figure 5.4: GAssist rules for the Global Statistics Model (Without MNPeano)

Chapter 6

Robustness analysis

6.1 Chapter overview

As the results in the previous chapter provide a solid base to identify the curve to which a graph belongs (as long as MPeano and MNPeano are not presented simultaneously), our research advocates now to the practical uses of these findings.

This chapter presents a series of models, building up from the Global Statistics model, that redefine mainly the way in which to prepare the data, so as to take advantage of the characteristics of the attributes proposed.

By means of partitioning the coordinates in a graph, first in ordered groups and then in random order, and ultimately adding noise to the information available, we will reach a solid model that surpasses the boundaries of our studied curves, and allows for a novel way of finding solutions to the Traveling Salesman Problem.

The rest of the chapter presents the different models, maintaining the structure of description, experimentation proposal, and experimental results.

6.2 Partition Statistics

6.2.1 Description

The Global Statistics model succeeded at classifying a curve 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 continuous section of it. Since the ideas presented so far lack a way to deal with said situation, the concept of *Partitions* was devised. We will call **Partition** a

continuous subset of the coordinates from one of the curves.

The **Partition Statistics** model implies partitioning the FTSP's and using said partitions to create GAssist instances with the attributes used so far. Previous to the construction details, we must mention that the attribute for *Number of Cities* is omitted from this model onwards, since the partitioning of the curves in the way described implies that not even the partitions for the same order of the same curve share this value.

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 6.1 is applied to our model.

```

partitionsLeft: Integer
partitions ← an empty collection
CoordinatesLeft ← curve.coordinates.size
from ← 1
to ← 0
WHILE (partitionsLeft > 0)
    coordinatesInPartition ← Round( $\frac{\textit{CoordinatesLeft}}{\textit{partitionsLeft}}$ )
    to ← min(to +
        coordinatesInPartition, curve.coordinates.size)
    partitionCoordinates ← curve.coordinates.copy(
        from, to)
    partitions.add( partitionCoordinates )
    from ← to + 1
    partitionsLeft ← partitionsLeft - 1
RETURN partitions

```

Algorithm 6.1: Instance Partitioning

As an example we will consider the first order of Koch Tour and David Tour. Coordinates are presented in table 6.1.

For this example we will consider 5 partitions per curve. Tables 6.2 and 6.3 show the result of applying the algorithm presented.

Each partition can now be considered as an independent graph, which means each can be converted into a GAssist instance as described for the Global Statistics model.

Koch Tour			David Tour					
#	X	Y	#	X	Y	#	X	Y
1	-15	-9	1	-10	18	13	-10	-18
2	-10	0	2	-5	27	14	-20	-18
3	-15	9	3	5	27	15	-25	-9
4	-5	9	4	10	18	16	-20	0
5	0	18	5	20	18	17	-25	9
6	5	9	6	25	9	18	-20	18
7	15	9	7	20	0			
8	10	0	8	25	-9			
9	15	-9	9	20	-18			
10	5	-9	10	10	-18			
11	0	-18	11	5	-27			
12	-5	-9	12	-5	-27			

Table 6.1: Koch Tour and David Tour - Order 1 Coordinates

Partition	Included Coordinates
1	(0,0), (5,9)
2	(0,18), (10,18), (15,27)
3	(20,18), (30,18)
4	(25,9), (30,0), (20,0)
5	(15,-9), (10,0)

Table 6.2: Partitions for the first order of Koch Tour considering 5 partitions

Partition	Included Coordinates
1	(0,0), (5,9), (15,9), (20,0)
2	(30,0), (35,-9), (30,-18), (35,-27)
3	(30,-36), (20,-36), (15,-45)
4	(5,-45), (0,-36), (-10,-36), (-15,-27)
5	(-10,-18), (-15,-9), (-10,0)

Table 6.3: Partitions for the first order of David Tour considering 5 partitions

Folding

With 5 partitions for each of the 6 orders from the 4 curves, we have a total of 120 partitions. Our objective is to perform a 10-fold cross-validation with these instances, requiring a distribution of said instances into ten groups, ensuring an even distribution of the possible classes amongst the folds. To accomplish the distribution, we propose the use of Algorithm 6.2.

```

1) a) Start with an empty collection C
   b) for order = 1...6
       add all the partitions from David Tour at order to C
   c) for order = 1...6
       add all the partitions from Koch Tour at order to C
   d) for order = 1...6
       add all the partitions from MPeano at order to C
   e) for order = 1...6
       add all the partitions from MNPeano at order to C

2) a) for each partition p in C
       Let f be (number of partitions considered) modulo
               (number of foldings)
       add p to fold f

```

Algorithm 6.2: Partition Folding

This procedure ensures an even distribution of the partitions, avoiding as best as possible the presence of more than 1 partition for each order of each curve. Table 6.4 shows the result of applying our algorithm considering 5 partitions. Elements in the table are indicated as C_{op} where C is the code for the curve (**D**avid Tour, **K**ock Tour, **M**PEano, **MN**PEano), o is the order, and p is the partition number. Each of these 10 folds present 12 instances, 3 from each curve.

Fold	Partitions											
1	D_{11}	D_{31}	D_{51}	K_{11}	K_{31}	K_{51}	M_{11}	M_{31}	M_{51}	N_{11}	N_{31}	N_{51}
2	D_{12}	D_{32}	D_{52}	K_{12}	K_{32}	K_{52}	M_{12}	M_{32}	M_{52}	N_{12}	N_{32}	N_{52}
3	D_{13}	D_{33}	D_{53}	K_{13}	K_{33}	K_{53}	M_{13}	M_{33}	M_{53}	N_{13}	N_{33}	N_{53}
4	D_{14}	D_{34}	D_{54}	K_{14}	K_{34}	K_{54}	M_{14}	M_{34}	M_{54}	N_{14}	N_{34}	N_{54}
5	D_{15}	D_{35}	D_{55}	K_{15}	K_{35}	K_{55}	M_{15}	M_{35}	M_{55}	N_{15}	N_{35}	N_{55}
6	D_{21}	D_{41}	D_{61}	K_{21}	K_{41}	K_{61}	M_{21}	M_{41}	M_{61}	N_{21}	N_{41}	N_{61}
7	D_{22}	D_{42}	D_{62}	K_{22}	K_{42}	K_{62}	M_{22}	M_{42}	M_{62}	N_{22}	N_{42}	N_{62}
8	D_{23}	D_{43}	D_{63}	K_{23}	K_{43}	K_{63}	M_{23}	M_{43}	M_{63}	N_{23}	N_{43}	N_{63}
9	D_{24}	D_{44}	D_{64}	K_{24}	K_{44}	K_{64}	M_{24}	M_{44}	M_{64}	N_{24}	N_{44}	N_{64}
10	D_{25}	D_{45}	D_{65}	K_{25}	K_{45}	K_{65}	M_{25}	M_{45}	M_{65}	N_{25}	N_{45}	N_{65}

Table 6.4: Partitions Statistics 10-Fold - 5 Partitions

6.2.2 Experimental Results

In order to analyze the usefulness of the presented model in classifying partitions of the curves, the algorithms presented were used to generate data for 2 to 10

partitions, considering all 4 curves from order 1 to 6. Although considering 1 partition would have been interesting since it would represent the previous model, it was not part of the mentioned experiments since only 24 instances would have been available, making it impossible to prepare the 10 folds with at least 1 instance of each curve. No more that 10 partitions were considered since the first order of the curves has so a small number of coordinates that it would have meant reducing the curve beyond sense. Figure 6.1 presents the results obtained.

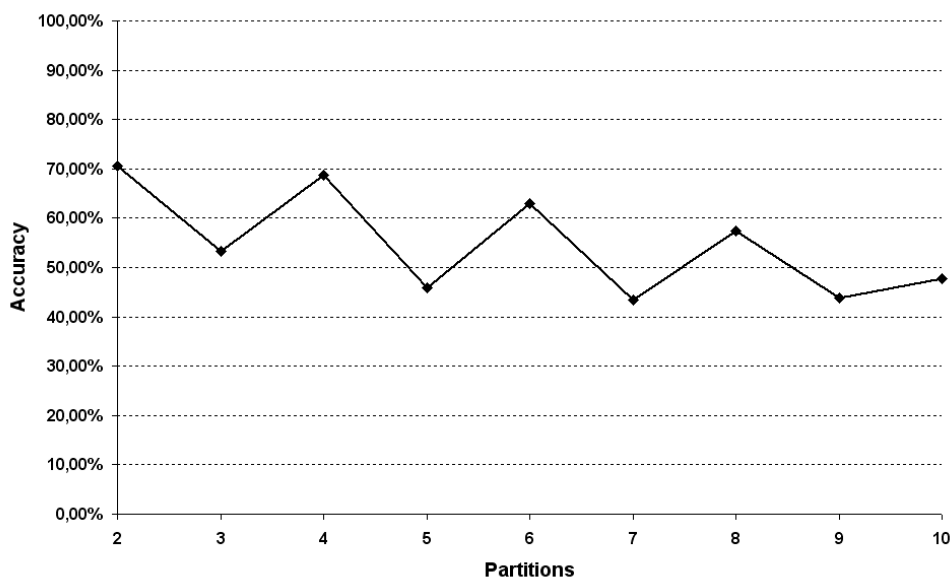


Figure 6.1: Partitions Model Accuracy (4 Curves, 6 Orders)

Based on the idea that the higher orders provide a more distinct shape of the curves, the same experiment was 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 6.2 shows a similar tendency in the results, although with a higher average.

Since it was mentioned that an even distribution of the curves amongst the folds is important, figures 6.3 and 6.4 show the individualization of the results from figure 6.2, considering in the first case the accuracies for the contexts where there was an equal number of instances for each curve in each fold, and in the second the rest of the contexts.

Before drawing conclusions from these results, 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 be-

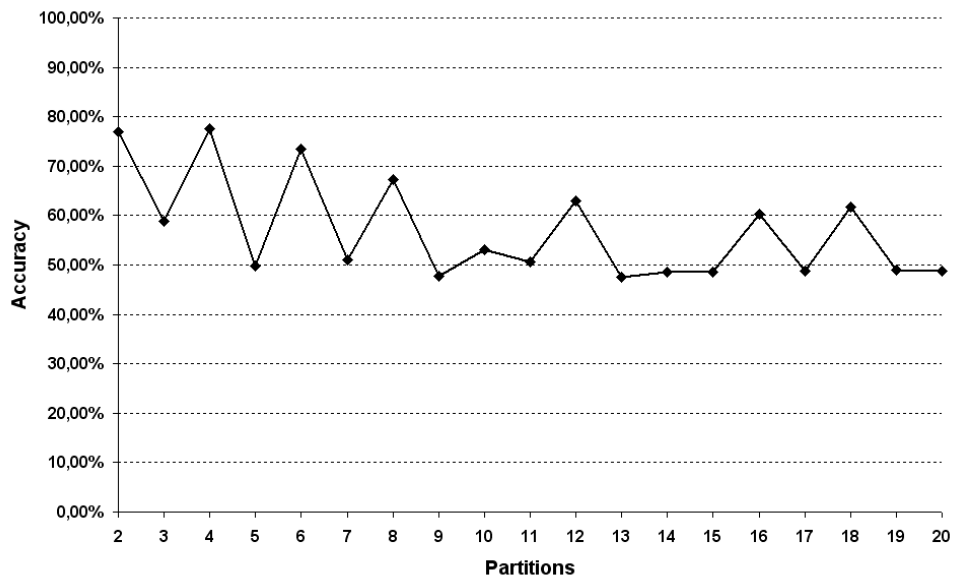


Figure 6.2: Partitions Model Accuracy (4 Curves, 5 Orders)

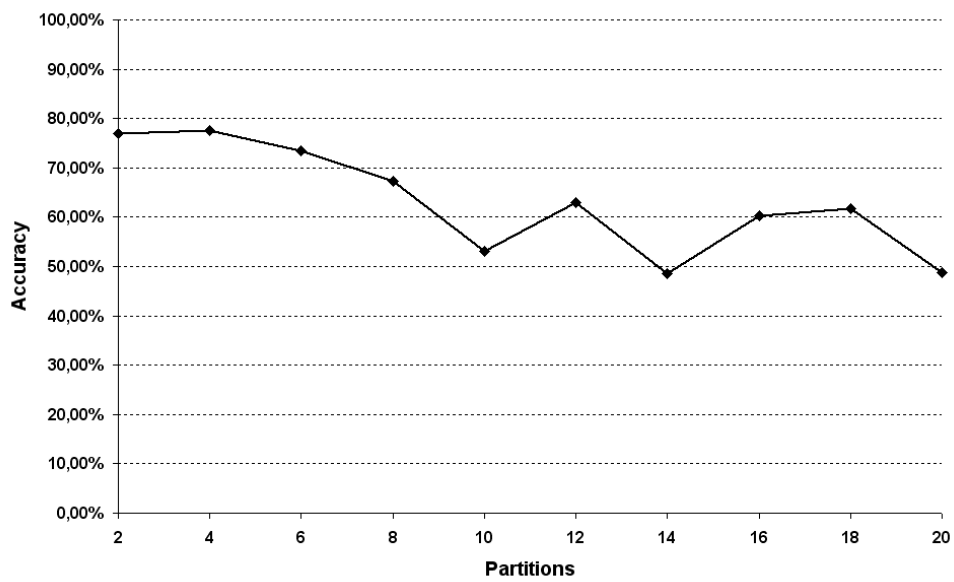


Figure 6.3: Partitions Model Accuracy (4 Curves, 5 Orders, Exact Distribution)

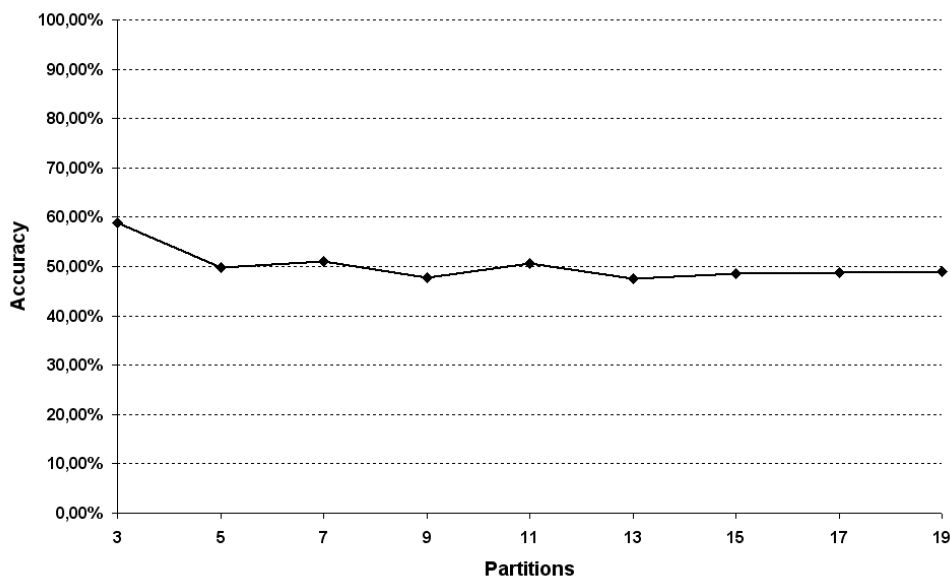


Figure 6.4: Partitions Model Accuracy (4 Curves, 5 Orders, Inexact Distribution)

tween MPeano and MNPeano, which is thought to confuse GAssist. Results are presented in Figures 6.5, 6.6, 6.7 and 6.8

6.2.3 Conclusions

The last figures show that the Partition Statistics model behaves quite well once the indecision caused by MNPeano is removed. Also, we can conclude that an increase in the number of partitions, and consequently a reduction in the number of coordinates in each, reduces the accuracy of our model. In order to be of use, we understand that a compromise is to be reached between the partitioning of the curves to obtain a larger number of instances for the data sets, and the objective accuracy intended for the model. All of the figures presented so far share the same boundaries in the values shown, so as to allow for a better comparison between them. It is thanks to this presentation that it can be easily seen that the theory about the importance of the exact distribution of the instances holds true.

One of the great benefits of using GAssist as our learning classifier system is that the rules it uses to classify are clear to a human reader. This quality will be of help now, since we can compare the rules obtained when running the experiments with and without MNPeano, to increase our understanding of the differences in accuracy. Figure 6.9 contains the rules for the 8th run of Fold 2, in the experiment with all curves, from orders 2 to 6, considering 4 partitions; this run got a 100% testing accuracy. The ruleset shows an intricate behaviour when trying to distinguish between MPeano and MNPeano, whereas the rules for David

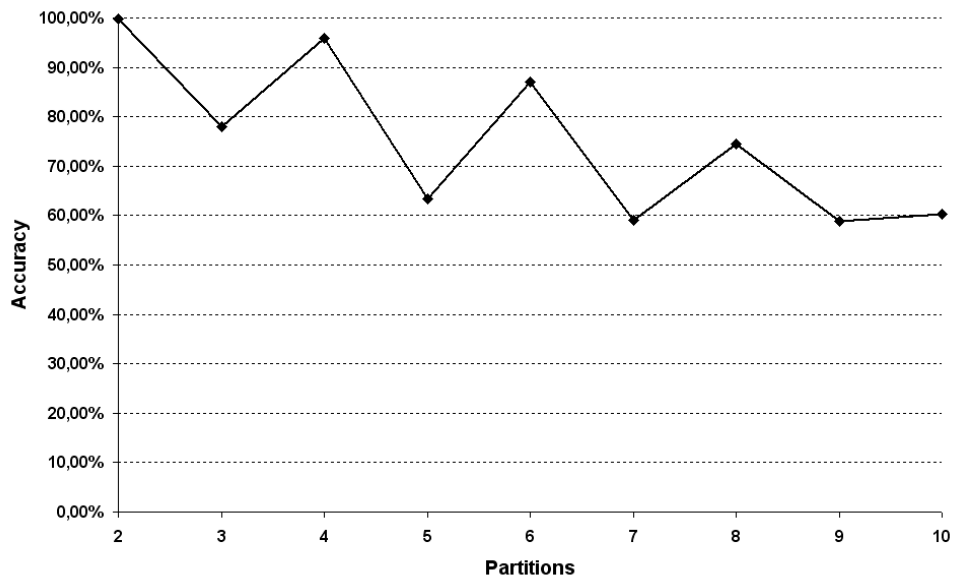


Figure 6.5: Partitions Model Accuracy (3 Curves, 6 Orders)

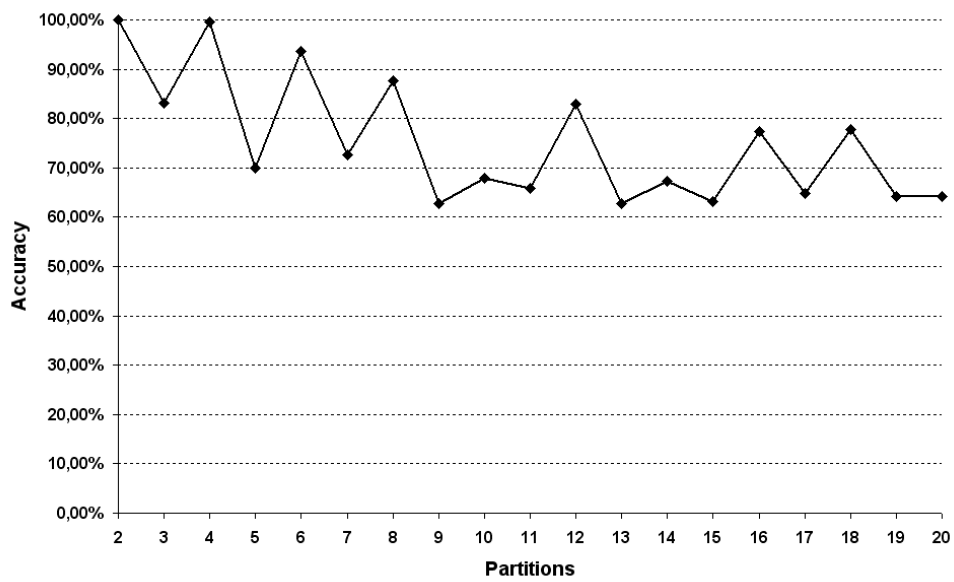


Figure 6.6: Partitions Model Accuracy (3 Curves, 5 Orders)

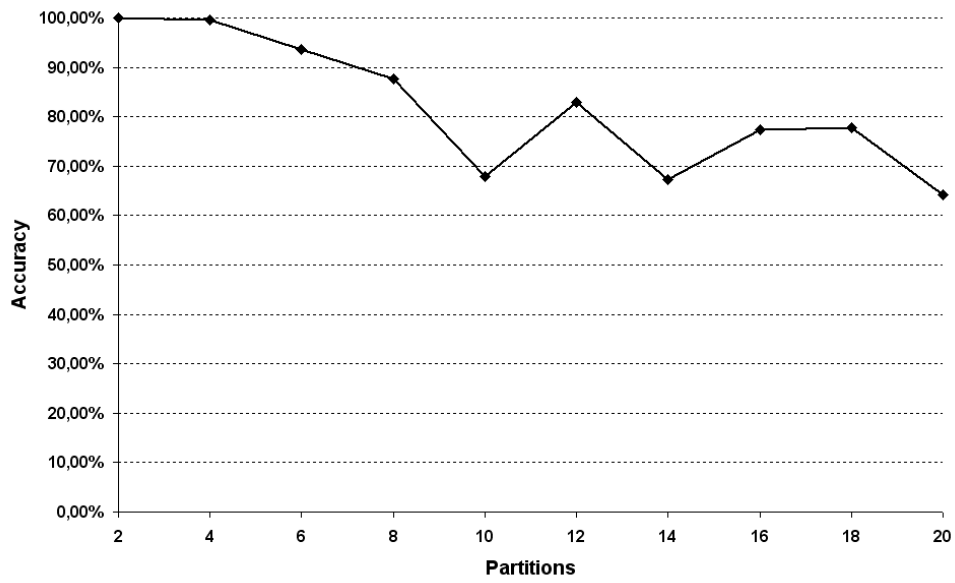


Figure 6.7: Partitions Model Accuracy (3 Curves, 5 Orders, Exact Distribution)

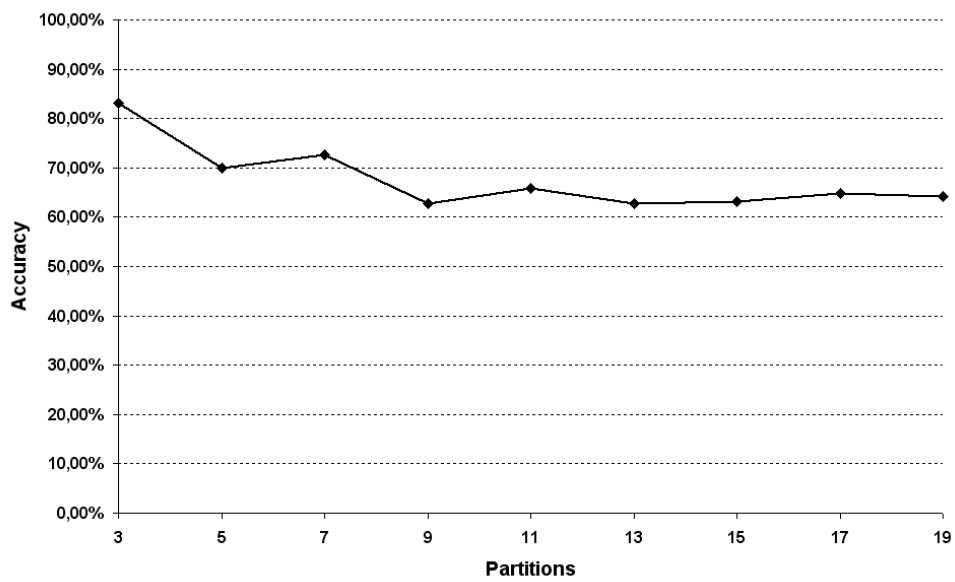


Figure 6.8: Partitions Model Accuracy (3 Curves, 5 Orders, Inexact Distribution)

Tour and Koch Tour are quite simpler. On the other hand, figure 6.10 shows the ruleset for the experiment without MNPeano, more specifically the 10th run for Fold 1, but in this case considering 8 partitions. Apart from the importance of the average accuracy of the category this experiment belongs to, it must be remarked that the 100% testing accuracy obtained by this run has greater value than the one presented previously for all curves, because since it worked with 8 partitions it means that lesser coordinates in a partition were required, which means that the ruleset provided in figure 6.10 would be more useful in classifying part of a graph with unknown class.

```

0:Att yAxisCenterAveragedSpread is
    [>0.5195120368075227] |
    DavidTour
1:Att averageNearestNeighbours is
    [<1.7036313657407407]
    [1.8884548611111105,1.9254195601851845] |
    MNPeano
2:Att maximumXcomponent is
    [>0.8634920634920635] |
    Att maximumYcomponent is
    [>0.8284149484536083] |
    Att yAxisCenterAveragedSpread is
    [>0.41679714297922293] |
    Att averageNearestNeighbours is
    [<1.8953857421875009]
    [1.957763671875001,1.9785563151041679] |
    MNPeano
3:Att maximumXcomponent is
    [>0.5904761904761905] |
    Att maximumYcomponent is
    [>0.5729438717067583] |
    MPeano
4:Att xAxisCenterAveragedSpread is
    [>0.5304640714476779] |
    DavidTour
5:Default rule -> KochTour

```

Figure 6.9: Partition Statistics GAssist Rules - 4 curves

6.2.4 Future Work

Encouraged by the results obtained in the previous section, we intend, as a follow up work from this thesis, to collect the rules with the best accuracies and experiment on their capacity to classify all of the partitions built during the ex-

```

0:Att maximumXcomponent is
  [0.45578231292516996,0.9659863945578235] |
  DavidTour
1:Att maximumYcomponent is
  [<0.45578231292516996]
  [>0.9659863945578235] |
  Att yAxisCenterAveragedSpread is
  [<0.45872076641307336]
  [>0.5351742274819189] |
  MPeano
2:Att xAxisCenterAveragedSpread is
  [>0.4792978195336551] |
  Att averageNearestNeighbours is
  [<1.9353298611111092]
  [>1.9670138888888868] |
  DavidTour
3:Att xAxisCenterAveragedSpread is
  [>0.4929920429489024] |
  MPeano
4:Att maximumYcomponent is
  [0.5714285714285714,0.7142857142857142] |
  Att averageNearestNeighbours is
  [>1.8878038194444449] |
  DavidTour
5:Default rule -> KochTour

```

Figure 6.10: Partition Statistics GAssist Rules - 3 curves

periments. The objective of this idea is to prove that these winning rule sets are general enough to succeed at the classification task of the FTSP's.

6.3 Sample Statistics

6.3.1 Description

The curves used so far respected the construction rules, and partitions were built with coordinates that belonged to the same region of a curve as a result of its construction. But, if we intend to put the results of this work to good use, we need to obtain a model that is robust enough to recognize a Koch Tour like graph when it sees one, even if some coordinates are missing or if half of them are some distance away from where they should be (as long as that noise doesn't impact on the optimum path that should be answered).

This is why we now present a model called **Sample Statistics**. It is much like

the Partitions Statistics model, except that the curves are split into groups where the coordinates are not necessarily from a continuous part of the graph. Aiming at proving the robustness of the GAssist attributes proposed, we collect **samples** from the curves, calling Sample to a non-continuous subset of the coordinates from one of the curves, obtained from random positions of the coordinates list.

As an example of sampling a curve, we present in table 6.5 the results of applying our random sampler to the first order of David Tour.

Sample	Included Coordinates
1	(-10,-18),(15,-45),(30,-18),(20,0)
2	(30,-36),(-15,-9),(15,9),(35,-9)
3	(-15,-27),(5,9),(0,0)
4	(-10,-36),(35,-27),(0,-36),(-10,0)
5	(20,-36),(5,-45),(30,0)

Table 6.5: Samples for the first order of David Tour considering 5 samples

Note that, as explained, the size of the samples matches the size of the partitions (refer to Table 6.3). Now that the size and the rest of the structure needed for the objective of comparing this model to the previous one are clear, we proceed to detailing the result of running the same set of experiments as before.

6.3.2 Experimental Results

Maintaining the order used for the Partitions Statistics model, we present first in figure 6.11 the results when considering all 4 curves, from orders 1 to 6. Then, figures 6.12, 6.13 and 6.14 show the results when omitting order 1, considering 2 to 20 samples, and then discriminating between the cases with an equal number of instances per sample from the ones with a different number.

Also, we repeat in figures 6.15, 6.16, 6.17 and 6.18 the idea from the previous model of analyzing the impact of removing MNPeano from the set of curves, to allow for a more clear interpretation of GAssist learning capabilities.

6.3.3 Conclusions

Although results for this model do not maintain the accuracy levels of the previous one, we can see that when omitting MNPeano and the first order of the curves, an accuracy above 80% is achieved when the number of samples is low. This proves that at some point the Sample Statistics model still has the chance of providing good rules. The structure considered, then, only needs some sample coordinates to deduce the underlying curve they belong to. This shows that the utility of the statistics proposed for the instances is a novel finding in itself.

It is now that the focus of this work shows promising possibilities in the use of GAssist as proposed to find optimum tours for TSP. Since these curves provide

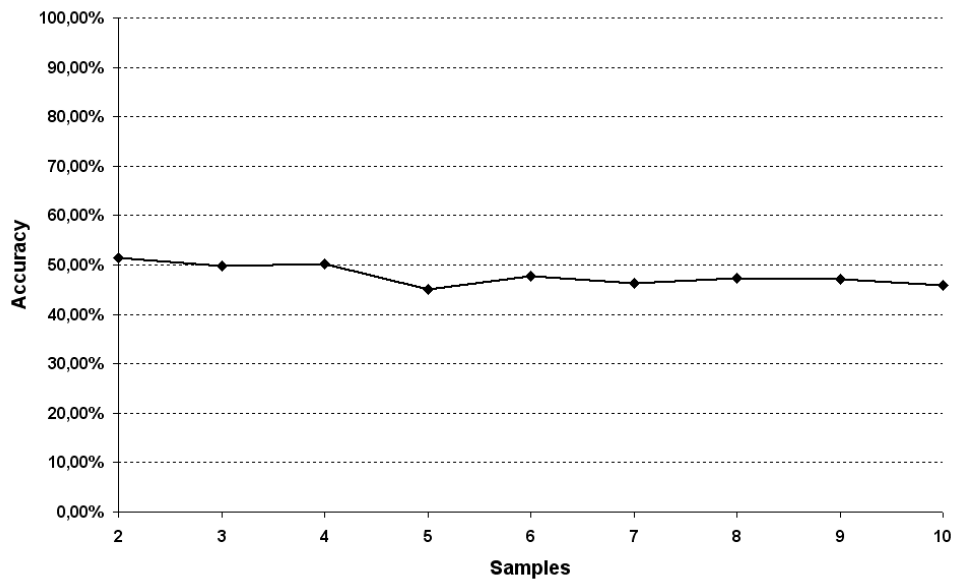


Figure 6.11: Samples Model Accuracy (4 Curves, 6 Orders)

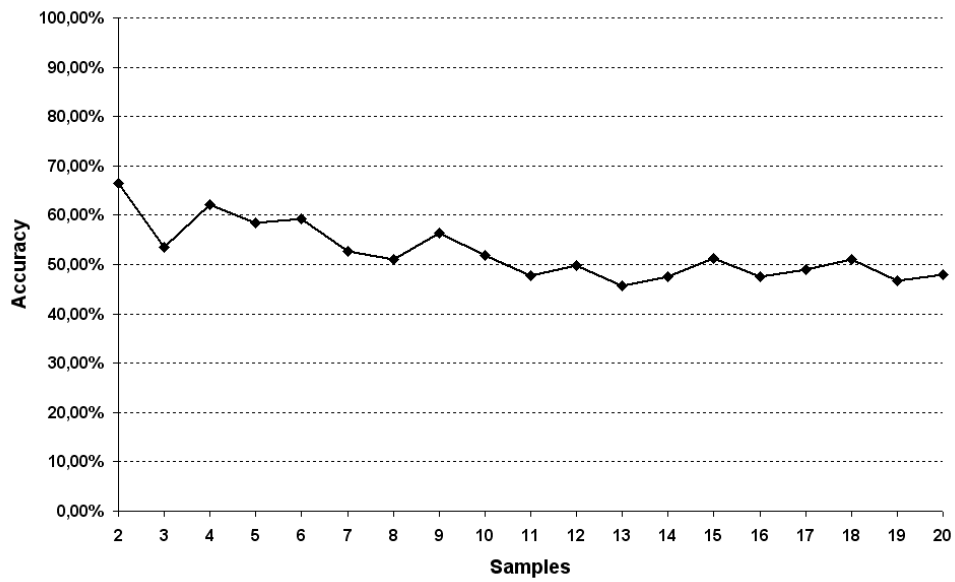


Figure 6.12: Samples Model Accuracy (4 Curves, 5 Orders)

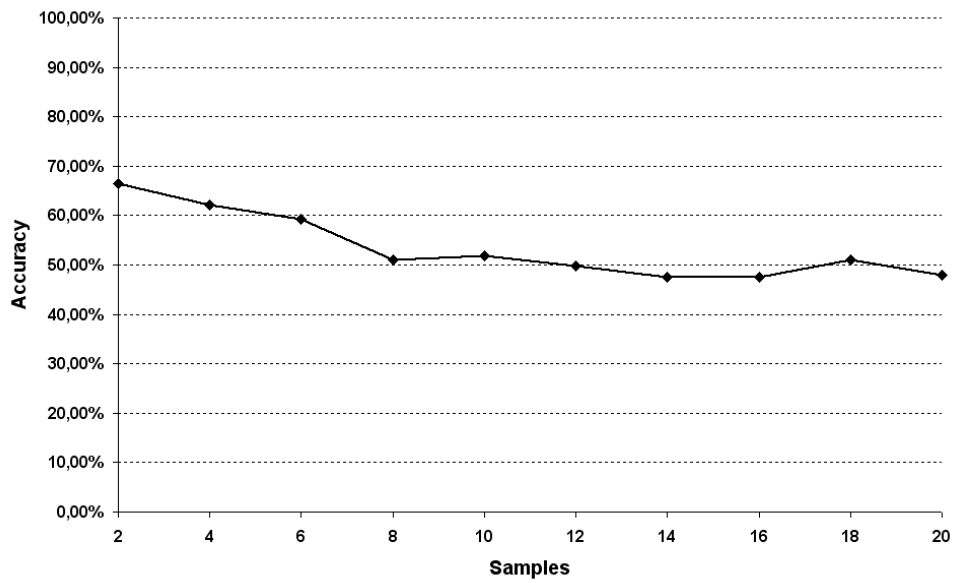


Figure 6.13: Samples Model Accuracy (4 Curves, 5 Orders, Exact Distribution)

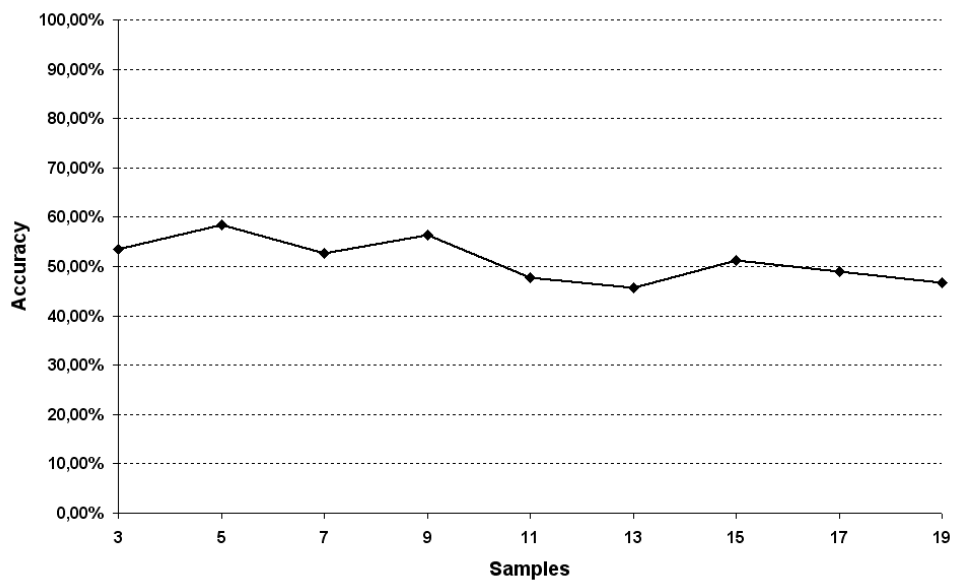


Figure 6.14: Samples Model Accuracy (4 Curves, 5 Orders, Inexact Distribution)

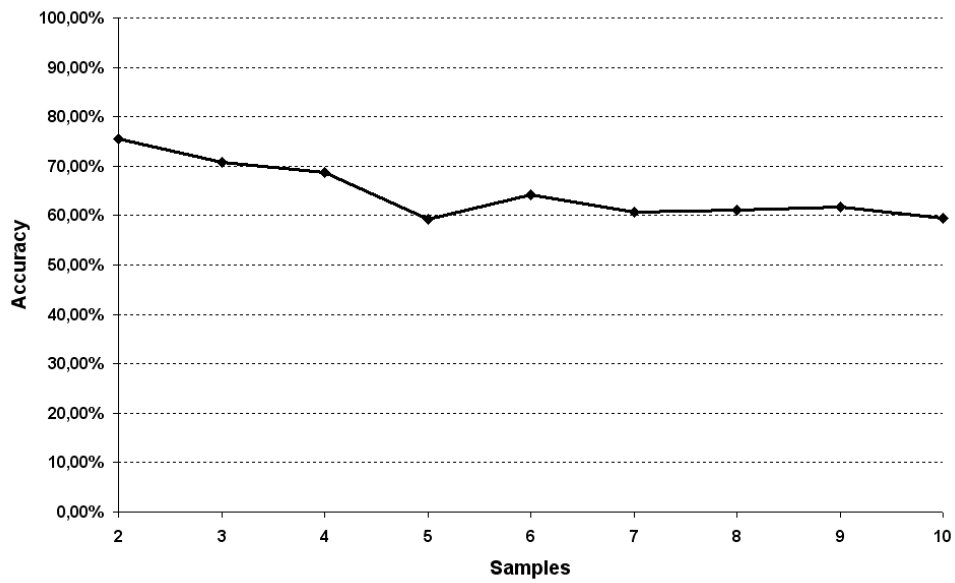


Figure 6.15: Samples Model Accuracy (3 Curves, 6 Orders)

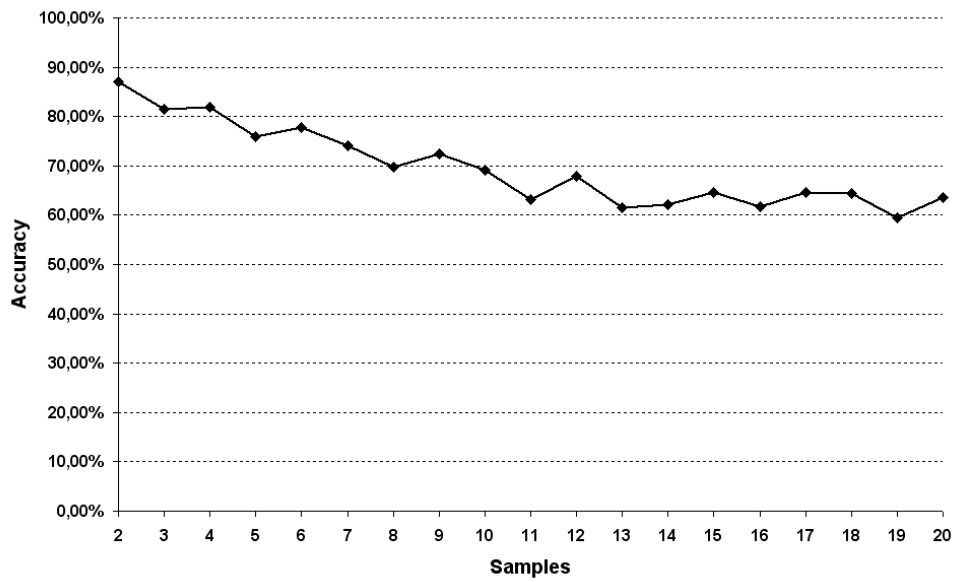


Figure 6.16: Samples Model Accuracy (3 Curves, 5 Orders)

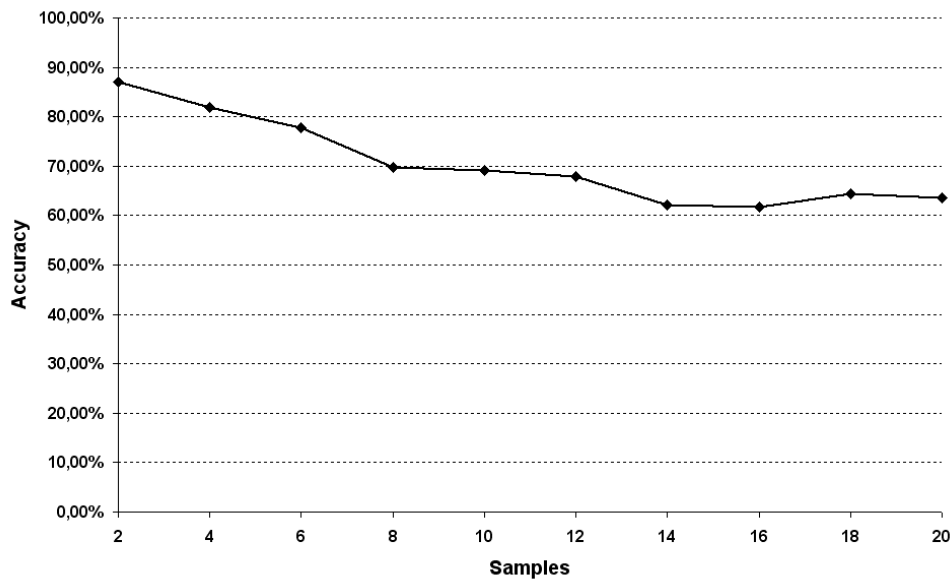


Figure 6.17: Samples Model Accuracy (3 Curves, 5 Orders, Exact Distribution)

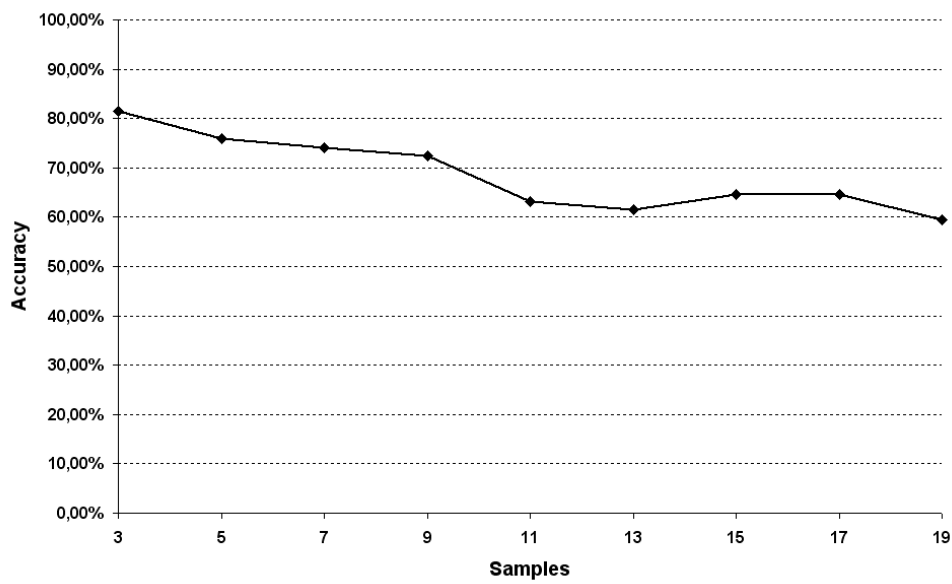


Figure 6.18: Samples Model Accuracy (3 Curves, 5 Orders, Inexact Distribution)

by construction a method to find their optimum path, we could take samples from any TSP and use GAssist to determine which of these curves it resembles most, to apply then the corresponding method. But before doing so, we need to test the tolerance of this ideas to a subtle movement of the coordinates from their expected place, something the next model will deal with.

6.4 Scrambled Statistics

6.4.1 Description

While the previous chapter proposed the features for the representation of TSP instances, this chapter has focused so far in working with the data available for the experiments, in order to analyze the robustness of the ideas presented. The **Scrambled Statistics** model continues this path, since its objective is to distort the position of the coordinates, and check how much noise our GAssist experiments can 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 a way in which to experiment with the amount of noise present and understand its effect on the features of the model.

Figure 6.19 shows the shape of the first order of David Tour, with the coordinates that make up the graph clearly shown. By computing the variance of the coordinates position in both axis, we can obtain a gaussian distribution based on it, with a mean of 0, and apply a white noise to the coordinates position using Algorithm 6.3.

```

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 y 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 6.3: Coordinate Scrambling

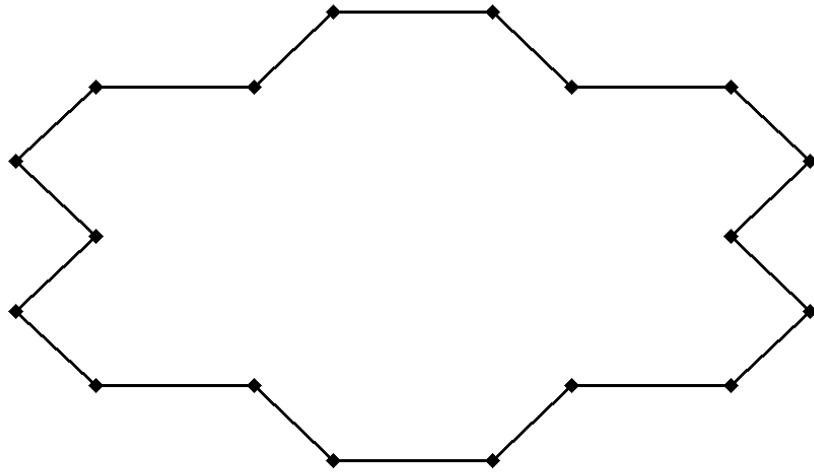


Figure 6.19: David Tour - Order 1 - No Scrambling

The matter lies then in choosing a scrambling ratio and observing what happens to the shape of the curve. If scrambling ratio is set to 0, figure 6.19 is obtained once again. But if we raise it to 5% (0.05), we obtain figure 6.20 instead. Increasing this value to 10% and 20% gives us figures 6.21 and 6.22.

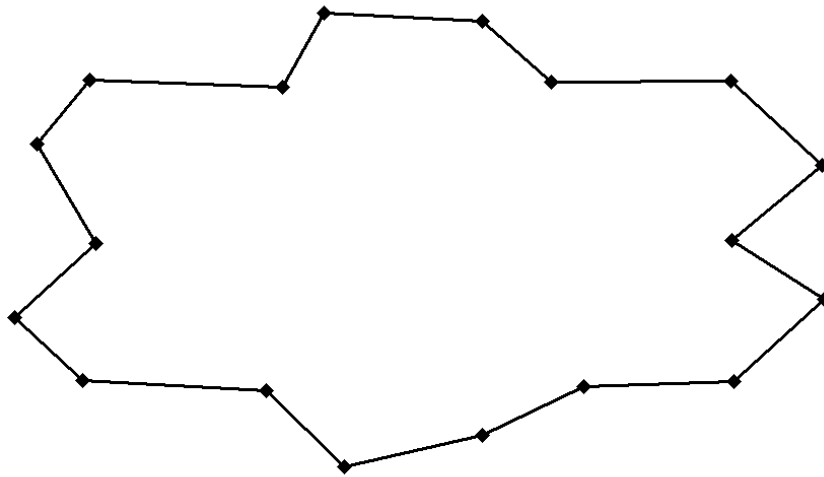


Figure 6.20: David Tour - Order 1 - 5% Scrambling

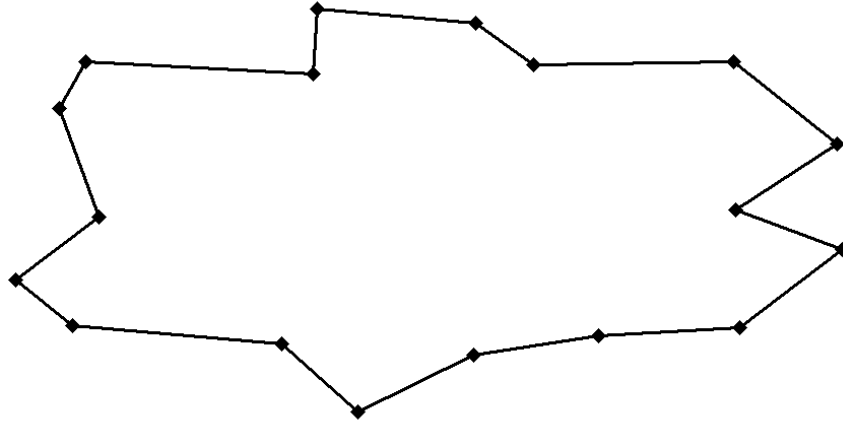


Figure 6.21: David Tour - Order 1 - 10% Scrambling

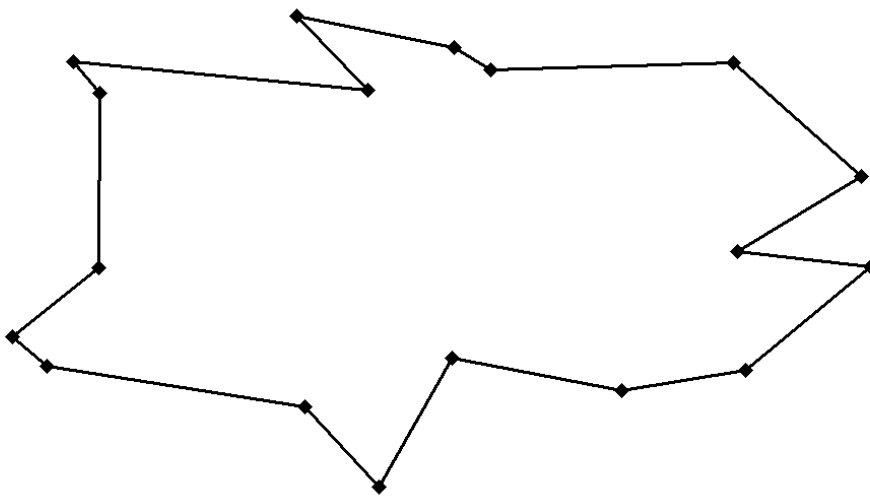


Figure 6.22: David Tour - Order 1 - 20% Scrambling

As can be seen from the figures presented, at 20% scrambling the graph is distorted enough to lose almost all resemblance to its original shape, and it is this reason that leads us to considering 20% an upper limit in our expectations for the possibilities of GAssist capacity to learn.

Considering that the Sample Statistics model gave lower accuracies than the Partition Statistics model before it, we will stick with the latter for our next experiments since we need to start with a known good value and see how well it stands the scrambling of the coordinates. 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.

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.

6.4.2 Experimental Results

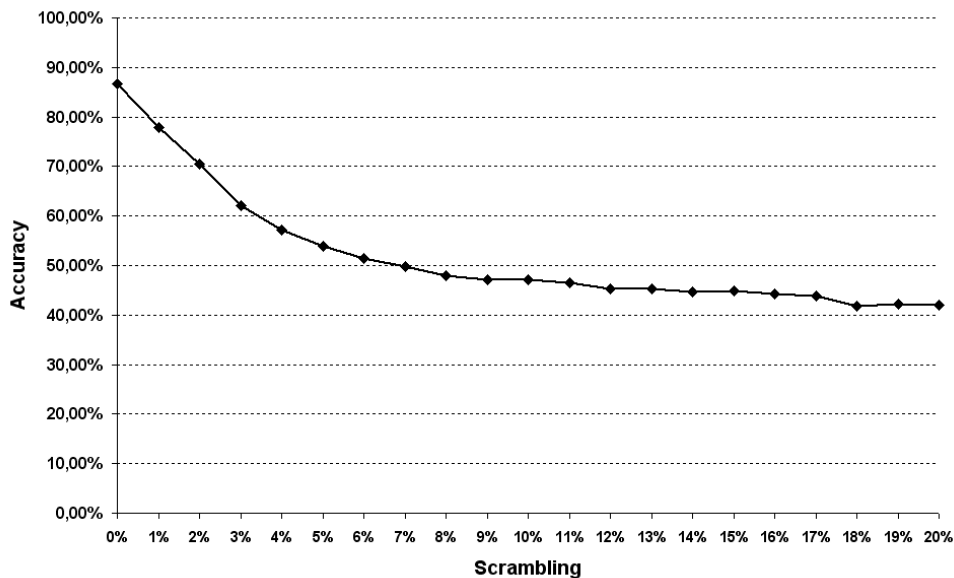


Figure 6.23: Scrambling Statistics Accuracy Results - 8 Partitions

In figures 6.23 and 6.24, we can see the real impact of scrambling the testing data in the partitions model. At 0%, the values lie where they were for the original model. As scrambling increases, accuracy drops steadily until it reaches values where no information is obtained (remember that because of the low number of classes, even a naive classifier would obtain a 33% accuracy).

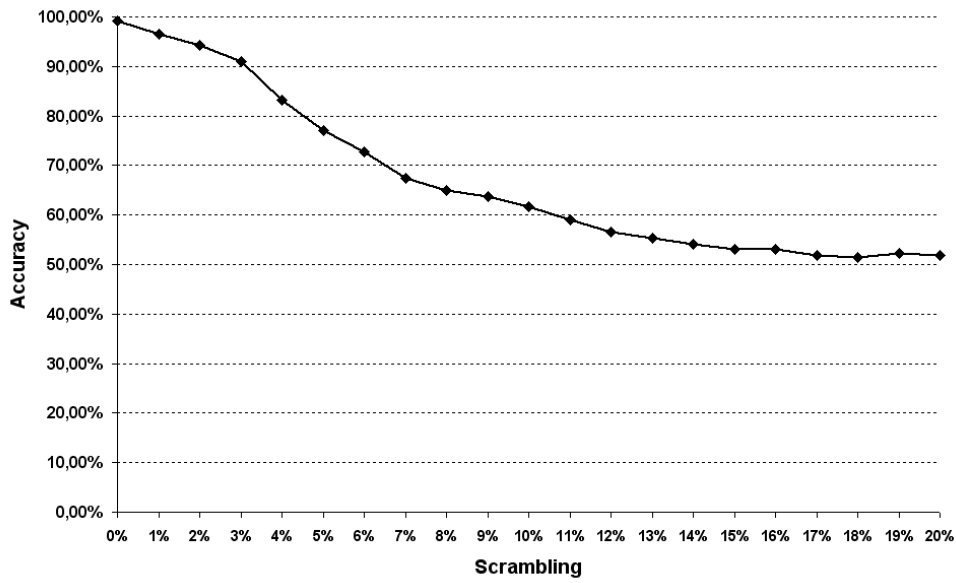


Figure 6.24: Scrambling Statistics Accuracy Results - 4 Partitions

6.4.3 Conclusions

Results show a promising sustaining of the high accuracies for scrambling below 5%. This confirms that the features proposed do not cause enough confusion to GAssist until they get significantly displaced from their original positions, proving the usefulness of the features presented in this thesis.

Chapter 7

From instance/edge classification to TSP optimisation: a proof of concept

7.1 Chapter overview

Up to now, we have analyzed the possibilities of using an LCS to learn in different scenarios information related to our TSP instances. This chapter wraps all of the previous research by presenting a simple program, composed of two sub-algorithms, that contains the knowledge gathered so far, and uses it to suggest a possible solution to a TSP instance.

This chapter is structured as follows. First we will review a way to take any of the rule sets obtained with the triplets model and use it to build a Hamiltonian Path in a graph. Then another program will be shown, which identifies to which of the TSP instances mentioned a graph belongs to. Finally we will bridge both ideas in a single system that will receive a group of coordinates from any order of any of the TSP instances and produce an interesting solution to the TSP.

7.2 Solving TSP with the rules obtained for the Triplets model

7.2.1 Objective

Our first objective will be to build, from a group of coordinates and a ruleset, a Hamiltonian cycle in the graph described by such coordinates. If the ruleset correctly allows to distinguish between good and bad edges (or triplets as in our

case) it can allow a program using it to iterate over all of the possible options and keep only those that make up the Hamiltonian cycle of minimum cost, which is to say, the solution to the TSP.

As the rule sets obtained in Chapter 4 work only for a given order of a given TSP instance, the system will need to be fed with the right rule set in order to produce interesting results. As a first step in obtaining a TSP solution, this section only aims at obtaining the solution knowing beforehand the best rule set for the task. We will call it, the *Good Enough Solution Finder*.

Algorithm description

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 set. It ideally ends up with a Hamiltonian cycle described by the triplets presented in Chapter 4.

Algorithm 7.1 shows the pseudocode for the *Good Enough Solution Finder*.

```

C: a set of coordinates
R: a rule set that indicates whether a given triplet is
    included or not in the solution to a TSP instance
S ← ∅    \\ This set of triplets can be interpreted as the edges
    that make up the solution to the TSP.
FOR (c ∈ C)
  IF (gradesS(c) < 2)
    IF (gradesS(c) = 1)
      e1 ← The edge ∈ S connecting c
    ELSE
      e1 ← A randomly chosen edge between c and another
        coordinate c' with gradesS(c') < 2, such that it does not
        imply a non Hamiltonian cycle
      e2 ← A randomly chosen edge ≠ e1, between c and another
        coordinate c'' with gradesS(c'') < 2, such that it does not
        imply a non Hamiltonian cycle
      IF (R classifies the triplet defined by (e1, e2) as included)
        S ← S ∪ {(e1, e2)}
RETURN S

```

Algorithm 7.1: Pseudocode for the Good Enough Solution Finder

Algorithm complexity

To ensure the usefulness of the proposed algorithm, we analyze next its computing time.

1. Let n be the number of coordinates received and r the number of rules in the ruleset obtained by GAssist.

2. The outermost cycle iterates all of the coordinates $\rightarrow O(n)$
3. Obtaining the current grade of a coordinate can be done in the worst case by iterating over a list of connections for each of the coordinates $\rightarrow O(n)$
4. Deciding whether an edge implies a non Hamiltonian cycle means verifying if its addition would cause a cycle in the current solution. The worst case again would lead to a full iteration of n steps. This means that assigning a value to \mathbf{e}_1 can be achieved in $O(n)$
5. For the same reason, the value for \mathbf{e}_2 is computed in $O(n)$
6. Computing the Triplets model features for a pair of edges to apply the rules is done in $O(r)$.
7. The previous analysis shows that the worst time presented by Algorithm 7.1 will be bound by $O(n \times (n + n + n + r)) \equiv O(n(3n + r)) \equiv O(n(n + r))$. Considering that $r < 10$ for all the rulesets considered, we can express the complexity as $O(n(n + 10)) \equiv O(n(n)) \equiv O(n^2)$

7.2.2 Limitations

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 7.1 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.

7.3 Identifying the curves with the rules obtained for the Global Statistics model

7.3.1 Objective

Just as the rules from Chapter 4 provided us with rule sets to be used in Algorithm 7.1, we will make use now of the rule sets obtained in Chapter 5.

As these rule sets classify a TSP instance into the different classes, we will present an *Instance Classifier* algorithm to take advantage of this and tell us to which TSP instance (MPeano/MNPeano, Koch Tour, David Tour) a set of coordinates belongs to, regardless of the order of such instance.

7.3.2 Algorithm description

Algorithm 7.2 describes the *Instance Classifier*.

```

C: a set of coordinates
R: a rule set that classifies a set of Global Statistics
    features into one of the TSP instances
F ← Global Statistics features as described in Chapter 5
    computed from C
I ← The classification returned by R when applied to F
RETURN I
    
```

Algorithm 7.2: Pseudocode for the Instance Classifier

Algorithm complexity

Once the rule sets are obtained, the computation of the features can be done in $O(n)$ (iterating the coordinates) and the application of the ruleset is done in constant time $O(1)$, meaning Algorithm 7.2 has a final complexity of $O(n)$.

7.4 A system to solve TSP

Just by using the ideas proposed in Section 7.3 we can go from a set of coordinates to the name of the TSP instance. By means of the results presented in [23] and [22], 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 Chapter 5 reached a 100% accuracy, this means that 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 with complete certainty and in linear time. This is thanks to the fact that Algorithm 7.2 has a complexity of $O(n)$.

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 of any TSP instance
2. Determines by means of Algorithm 7.2 the class of TSP instance
3. Uses a pre-built table to deduce from the number of coordinates the order of the TSP instance
4. Applies Algorithm 7.1 to provide the edges that belong to a Hamiltonian Cycle, as an approximation of the optimum solution to the TSP for the presented coordinates

The computing time is bound by the complexity of Algorithm 7.2, the cost of looking up the value in the pre-built table, and finally the application of Algorithm 7.1. This can be expressed as: $O(n) + O(1) + O(n^2) \equiv O(n^2)$.

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” [34][35] 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 specific subset of the input space.

7.4.1 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 7.1 was applied, since it was one of the many which got a 100% testing accuracy when it was created for Chapter 5.
- **Coordinates to order table:** Based on the instances considered, Table 7.1 was built into the system.
- **Triplet Classifying Rule Set:** In order to test the learning achieved by the rules created in Chapter 4, three different classification schemes were considered:
 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 choose, from the rule sets created in Chapter 4, that with the highest testing accuracy for the instance considered.
 3. **Ensemble of High Accuracy Rule Sets:** Considering the five rule sets with the highest testing accuracy, we perform a majority voting to decide whether a Triplet should be considered or not for the solution proposed. Note that this includes the rule set from the previous scheme.

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.

```

0:Att yAxisCenterAveragedSpread is [>0.5936526479750781] |
    MPeano
1:Att xAxisCenterAveragedSpread is
    [0.3928348909657322,0.4523364485981311] |
    KochTour
2:Att maximumXcomponent is [>0.9848484848484848] |
    MPeano
3:Default rule -> DavidTour
    
```

Figure 7.1: GAssist rule set used to classify the instances

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 7.1: Instance order based on number of coordinates

7.4.2 Experimental Results

Figures 7.2, 7.3 and 7.4 present the result of executing the system proposed with the configuration previously detailed. Below every image, a value representing the length of the solution is indicated.

7.4.3 Conclusions

The first thing to mention about the previous results is that the rule set applied (Figure 7.1) classified all inputs correctly. This means that every set of coordinates was correctly identified with its instance, allowing every one of them to be solved with the intended rule set from those selected.

The most remarkable detail of the results is that the difference between the tour length from the random coin classification and 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 ideas presented in this chapter provide proof that a more complex algorithm coupled with better features for GAssist can guide the creation of high performing solutions for the TSP.

7.4.4 Future Work

Some improvements to the ideas presented in this chapter are:

- Use of Constructive Heuristics: instead of using the rule sets as the only guiding element for the solution, and completing the graph with nearest neighbour computations, some parts of the presented system can be used as the initial graph from where a Constructive Heuristic could build up its solution.
- Normalization of coordinates in the Triplets and addition of topological and geometrical features for the coordinates in the Triplet. Some of these can be:
 - Nearest Neighbour
 - Next Nearest Neighbour
 - Delaunay Tessellation Neighbour
 - Gabriel Graph Neighbour
 - Relative Neighbourhood Graph Neighbour
 - Minimum Spanning Tree Neighbour
- Application of topological and geometrical features already used in similar tasks. See for example [32], [15].





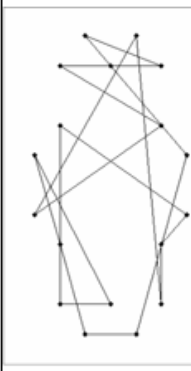
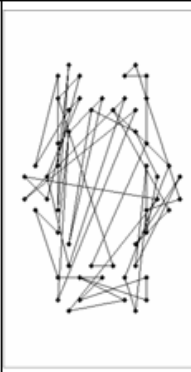
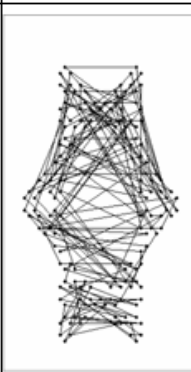

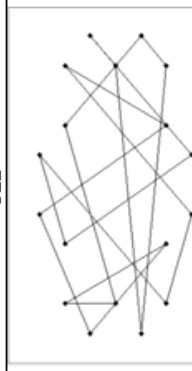
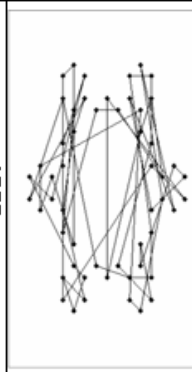
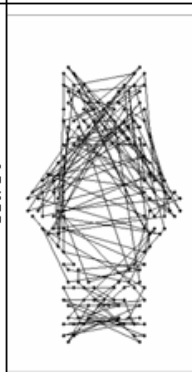
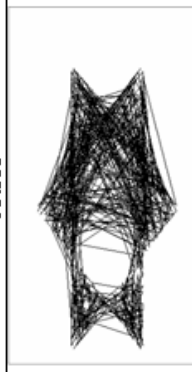
	Order 1	Order 2	Order 3	Order 4
Random Coin	 576	 3,625	 19,625	 126,017
Rule Set with High Accuracy	 522	 2,524	 16,784	 95,600
Ensemble of High Accuracy Rule Sets	 602	 2,287	 16,144	 87,577

Figure 7.2: David Tour results with the TSP Solving System for different rule sets


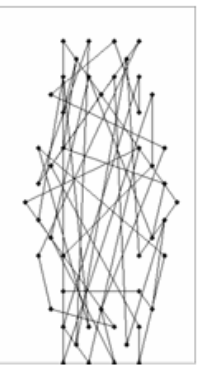
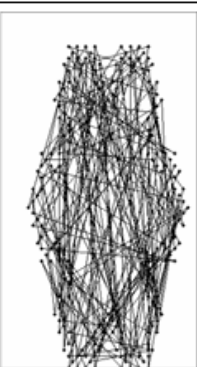

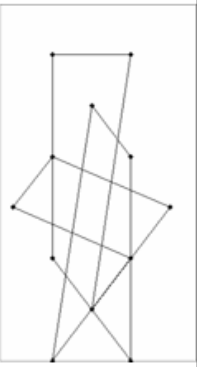
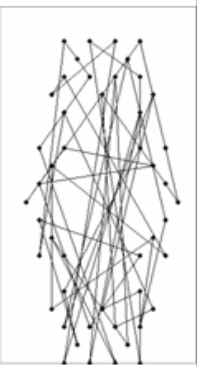


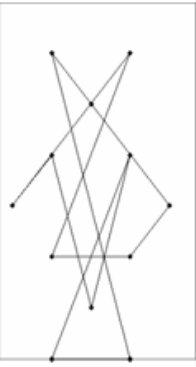



	Order 1	Order 2	Order 3	Order 4
Random Coin				
	221	2.537	31.589	379.504
Rule Set with High Accuracy				
	248	2.553	24.488	299.076
Ensemble of High Accuracy Rule Sets				
	242	2.233	23.559	294.526

Figure 7.3: Koch Tour results with the TSP Solving System for different rule sets

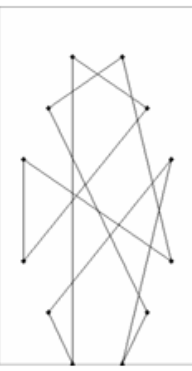
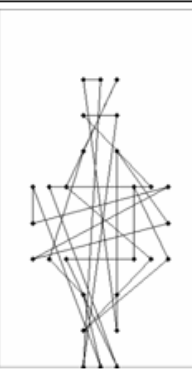
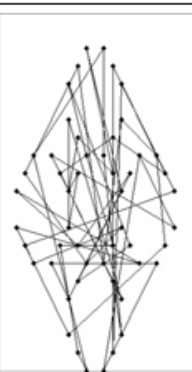
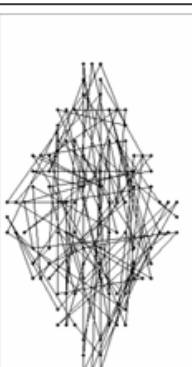
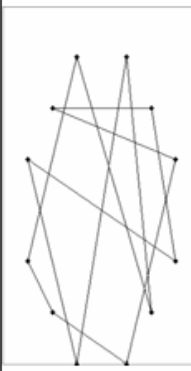
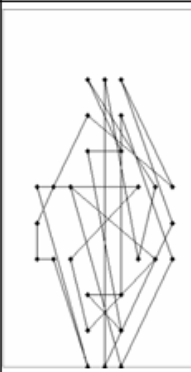


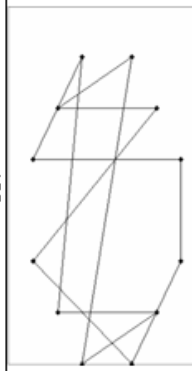
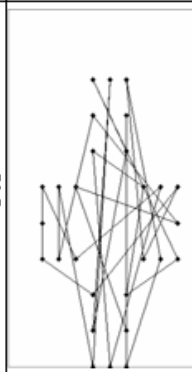
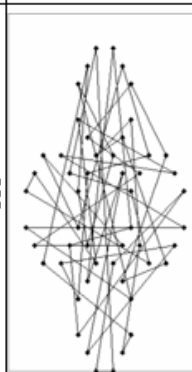
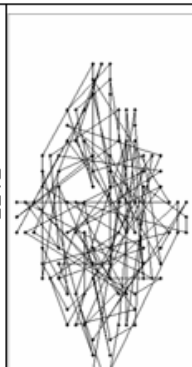
	Order 1	Order 2	Order 3	Order 4
Random Coin	 99	 379	 856	 2,902
Rule Set with High Accuracy	 107	 341	 868	 2,572
Ensemble of High Accuracy Rule Sets	 99	 369	 884	 2,603

Figure 7.4: MPeano results with the TSP Solving System for different rule sets

Chapter 8

Final Conclusions

8.1 Summary of what has been done

The objective of this thesis was to analyze the possibility of applying Learning Classifier Systems to Optimisation Problems. Throughout chapters 2 and 3, we summed up both the concept of LCS and that of the Travelling Salesman Problem, our optimization task of choice. Then chapters 4, 5 and 6 provided a set of ideas to face the task of solving TSP as a learning/classification challenge. Finally in Chapter 7 we went back to the initial objective of solving TSP by making use of the results previously obtained.

The findings and conclusions of this work are many. First we proved that even with simple features, a robust LCS like GAssist can provide meaningful results to interpret the structure of a graph problem like TSP. This encourages us to try to broaden our expectations on this topic, and look to other graph problems that may benefit from an approach similar to the one presented here. Also, we suggest graph features that have shown to discriminate between the instances chosen as test subjects, leaving room for the study of additional features when required to learn from a wider range of instances. With the final system proposed, we have presented a new way of solving TSP by adapting to different subsets of the input space, and obtaining an accuracy above average even without any optimization or in-depth analysis of the graph structures, but just by identifying that topological differences between the instances exist, and coding them into the features fed to the system.

Even different parts of the results obtained and the ideas presented can be combined with different already existing strategies to solving TSP, improving the overall value of the present work.

The features presented, apart from allowing the learning of rules that have a 100% accuracy when classifying, have shown to withstand low levels of scrambling,

and these results hold for special conditions when partitioning the graphs or presenting only samples of the total set of coordinates.

Additionally, the algorithms and formulas presented try to keep in line with the simplicity and clarity that is the basis for GAssist, allowing a better understanding of the processes and avoiding the confusion caused by complex layers of optimizations required for most state of the art solutions to NP problems. It is our belief that with this in mind, and by combining subsystems that give better results for different instances of a problem, a solid problem solver can be built, be it for TSP as in the case studied, or for any difficult task where the No Free Lunch Theorems present a theoretical limit when applying a straightforward and single method for any input.

8.2 Summary of what comes next

After analyzing the impact of the contents of this thesis up to the point where they were developed, we conclude this final chapter by reminding the reader the different aspects of the models and algorithms presented that can be further improved by researchers in the area:

8.2.1 Triplets

The Triplets model for finding parts of the TSP solution are based on the idea that there is more information in considering a path formed by 2 edges than just by taking a single edge. This concept can be further explored by considering longer paths. On the other hand, the opposite can be done, by considering only one edge, giving simpler features and algorithms, though this would require an analysis on the difference in performance achieved by such changes.

Also, as mentioned in Chapter 4, the coordinates could be normalized before used as input data, and additional information regarding the path and its components could be added. And as regards the training information, we suggest an improvement in the Triplets considered, as an increase in the number of “Not Included” elements would imply a closer representation of the ratio between the number of edges that are part of the optimum solution and the total number of possible edges in the graph.

8.2.2 Statistics Features

All of the results obtained in Chapters 5 and 6 are indeed interesting. Despite that, the robustness analysis proposed in Chapter 6 with respect to the features used shows that when the partitioning, size of sampling and scrambling becomes significant, the overall accuracy of GAssist to classify the instances drops at a steady rate. It would be interesting to find additional features that better

withstand these modifications to the input data, to allow for a more robust system that solves TSP.

With regards to the instances considered, additional instances would allow for a better understanding of the efficiency of the features selected. Also, the study of the differentiation between similar instances (such as MPeano and MNPeano in our case) constitutes an interesting area of research.

8.2.3 TSP Solving System

The system presented for solving TSP has a number of possible improvements that would allow it to be tested against the state of the art solutions present nowadays. As was mentioned during its description, we do not intend in this work to compare the results obtained with those achievable by a system specifically built for the task, but rather to understand the feasibility of our approach. It is now that its learning capabilities have been proved that improving it becomes a future area of work. We intend to follow on this path to add new topological features and combine the algorithms in the system with the ideas existing in state of the art algorithms.

Bibliography

- [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 and N. Krasnogor. Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO)*, pages 1441–1448. ACM Press New York, NY, USA, July 2006. ISBN 1-59593-186-4.
- [3] J. Bacardit, M. Stout, J.D. Hirst, J. Blazewicz, and N. Krasnogor. Coordination number prediction using learning classifier systems: performance and interpretability. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO)*, pages 247–254. ACM Press New York, NY, USA, 2006. ISBN 1-59593-186-4.
- [4] L. Bull. *Learning Classifier Systems: A Brief Introduction*. Springer-Verlag, 2004. pp. 3–14.
- [5] R.E. Burkard, V.G. Deineko, R. van Dal, J.A.A. van der Veen, and G.J. Woeginger. Well-solvable special cases of the tsp: A survey. Institute of Mathematics, University of Technology, Graz, Austria, 1995. manuscript.
- [6] E.K. Burke, M.R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *PPSN*, pages 860–869, 2006.
- [7] A.S. Fukunaga. Automated discovery of composite sat variable-selection heuristics. In *Eighteenth national conference on Artificial intelligence*, pages 641–648, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [8] A.S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 483–494, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.
- [9] M. Grotschel and G.L. Nemhauser. George dantzig’s contributions to integer programming. *Discrete Optimization*, 2007. Elsevier Science.

- [10] G. Gutin. Traveling salesman problems. *Handbook of Graph Theory*, 2003. CRC Press.
- [11] W.E. Hart, N. Krasnogor, and J.E. Smith, editors. *Recent advances in memetic algorithms*, volume 166 of *Studies in Fuzzyness and Soft Computing*. Springer Berlin Heidelberg New York, 2004.
- [12] J.H. Holland. Adaptation. *Progress in theoretical biology*, 4:263–293, 1976.
- [13] J.H. Holland and J.S. Reitman. Cognitive systems based on adaptive algorithms. In *Pattern-directed Inference Systems*. New York: Academic Press, 1978.
- [14] K.A. De Jong and W.M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the Twelfth International Conference on Artificial Intelligence IJCAI-91*, volume 2, 1991.
- [15] N. Krasnogor. Heurísticas rápidas para el tsp -2d euclideo y simétrico basadas en triangulación de delaunay y sus subgrafos (dissertation, licenciatura en informática, 5 years). Master’s thesis, Universidad Nacional de La Plata, La Plata, Buenos Aires, Argentina.
- [16] N. Krasnogor. Self-generating metaheuristics in bioinformatics: the protein structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181–201, 2004.
- [17] N. Krasnogor and S. Gustafson. A study on the use of “self-generation” in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.
- [18] N. Krasnogor and J.E. Smith. A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.
- [19] A. Mariano, P. Moscato, and M.G. Norman. Arbitrarily large planar tsp instances with known optimal tours. *Pesquisa Operacional*, 15:89–96, 1995.
- [20] A. Mariano, P. Moscato, and M.G. 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 November 1995.
- [21] K. Menger. Das botenproblem. *Ergebnisse Eines Mathematischen Kolloquiums*, 2:11–12, 1932.
- [22] P. Moscato and M.G. 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.
- [23] M.G. Norman and P. Moscato. The euclidean traveling salesman problem and a space-filling curve. *Chaos, Solitons and Fractals*, 6:389–397, 1995.
- [24] C.H. Papadimitriou. The euclidean traveling salesman problem is np-complete. *Theoret. Comput. Sci.*, 4:237–244, 1977.

- [25] H.O. Peitgen, H. Jürgens, and D. Saupe. *Chaos and Fractals*. Springer, February 2004.
- [26] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.
- [27] G. Rozenberg and A. Salomaa. *Mathematical Theory of L Systems*. Academic Press, Inc., Orlando, FL, USA, 1980.
- [28] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.
- [29] J.E. Smith. Co-evolving memetic algorithms: A review and progress report. *IEEE Transactions in Systems, Man and Cybernetics, part B*, 37(1):6–17, 2007.
- [30] M. Stout, J. Bacardit, J.D. Hirst, J. Blazewicz, and N. Krasnogor. Prediction of residue exposure and contact number for simplified hp lattice model proteins using learning classifier systems. In *Proceedings of the 7th International FLINS Conference on Applied Artificial Intelligence*, pages 601–608. World Scientific, August 2006.
- [31] M. Stout, J. Bacardit, J.D. Hirst, N. Krasnogor, and J. Blazewicz. From hp lattice models to real proteins: coordination number prediction using learning classifier systems. In *4th European Workshop on Evolutionary Computation and Machine Learning in Bioinformatics*, volume 3907 of *Springer Lecture Notes in Computer Science*, pages 208–220, Budapest, Hungary, April 2006. Springer. ISBN 978-3-540-33237-4.
- [32] 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), 2007.
- [33] S.W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.
- [34] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.
- [35] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

List of Algorithms

2.1	L-System Pseudocode	8
2.2	L-System Example	9
2.3	Fibonacci length strings L-System	9
2.4	Koch Tour L-System	10
2.5	David Tour L-System	10
2.6	MPeano L-System	11
2.7	MNPeano L-System	12
6.1	Instance Partitioning	50
6.2	Partition Folding	52
6.3	Coordinate Scrambling	65
7.1	Pseudocode for the Good Enough Solution Finder	72
7.2	Pseudocode for the Instance Classifier	74

List of Figures

2.1	Koch Tour Coordinates - Orders 0 to 5	13
2.2	Koch Tour Exact Solutions - Orders 0 to 5	14
2.3	David Tour Coordinates - Orders 0 to 5	15
2.4	David Tour Exact Solutions - Orders 0 to 5	16
2.5	MPeano Coordinates - Orders 0 to 5	17
2.6	MPeano Exact Solutions - Orders 0 to 5	18
2.7	MNPeano Coordinates - Orders 0 to 5	19
2.8	MNPeano Exact Solutions - Orders 0 to 5	20
4.1	Koch Tour - Order 1	32
4.2	Triplets Model - Testing Accuracy	37
4.3	GAssist rules for the 5 th run of Koch Tour Order 6 on group 1	38
5.1	Koch Tour Order 1 Nearest Neighbors	42
5.2	GAssist rules for the Global Statistics Model (With MPeano and MN- Peano)	46
5.3	GAssist rules for the Global Statistics Model (Without MPeano)	47
5.4	GAssist rules for the Global Statistics Model (Without MNPeano)	47
6.1	Partitions Model Accuracy (4 Curves, 6 Orders)	53
6.2	Partitions Model Accuracy (4 Curves, 5 Orders)	54
6.3	Partitions Model Accuracy (4 Curves, 5 Orders, Exact Distribution)	54
6.4	Partitions Model Accuracy (4 Curves, 5 Orders, Inexact Distribution)	55
6.5	Partitions Model Accuracy (3 Curves, 6 Orders)	56
6.6	Partitions Model Accuracy (3 Curves, 5 Orders)	56
6.7	Partitions Model Accuracy (3 Curves, 5 Orders, Exact Distribution)	57
6.8	Partitions Model Accuracy (3 Curves, 5 Orders, Inexact Distribution)	57
6.9	Partition Statistics GAssist Rules - 4 curves	58
6.10	Partition Statistics GAssist Rules - 3 curves	59
6.11	Samples Model Accuracy (4 Curves, 6 Orders)	61

6.12	Samples Model Accuracy (4 Curves, 5 Orders)	61
6.13	Samples Model Accuracy (4 Curves, 5 Orders, Exact Distribution)	62
6.14	Samples Model Accuracy (4 Curves, 5 Orders, Inexact Distribution)	62
6.15	Samples Model Accuracy (3 Curves, 6 Orders)	63
6.16	Samples Model Accuracy (3 Curves, 5 Orders)	63
6.17	Samples Model Accuracy (3 Curves, 5 Orders, Exact Distribution)	64
6.18	Samples Model Accuracy (3 Curves, 5 Orders, Inexact Distribution)	64
6.19	David Tour - Order 1 - No Scrambling	66
6.20	David Tour - Order 1 - 5% Scrambling	66
6.21	David Tour - Order 1 - 10% Scrambling	67
6.22	David Tour - Order 1 - 20% Scrambling	67
6.23	Scrambling Statistics Accuracy Results - 8 Partitions	68
6.24	Scrambling Statistics Accuracy Results - 4 Partitions	69
7.1	GAssist rule set used to classify the instances	76
7.2	David Tour results with the TSP Solving System for different rule sets	78
7.3	Koch Tour results with the TSP Solving System for different rule sets	79
7.4	MPeano results with the TSP Solving System for different rule sets	80

List of Tables

2.1	Fibonacci length strings L-System. Orders 0 to 8	9
4.1	Attributes for an instance of the Triples model	30
4.2	Attributes for an instance of the Triples model - Reverse	30
4.3	Attributes for an instance of the Triples model - Permutations	31
4.4	Triplets model Attributes - Koch Tour - Order 1	31
4.5	Triplets model Instances - Koch Tour - Order 1	32
4.6	Triplets model Instances - Koch Tour - Order 1 - Reverse	33
4.7	Randomly generated instances not present in Koch Tour order 1	34
4.8	Triplets model Groups - Koch Tour - Order 1	34
4.9	Triplets model Testing Accuracy Results - Koch Tour - Order 1	35
5.1	Global Statistics Centered Coordinates Example	41
5.2	Global Statistics Spread Example	41
5.3	Global Statistics Testing Accuracy	44
5.4	Global Statistics Without MPeano Testing Accuracy	45
5.5	Global Statistics Without MNPeano Testing Accuracy	46
6.1	Koch Tour and David Tour - Order 1 Coordinates	51
6.2	Partitions for the first order of Koch Tour considering 5 partitions	51
6.3	Partitions for the first order of David Tour considering 5 partitions	51
6.4	Partitions Statistics 10-Fold - 5 Partitions	52
6.5	Samples for the first order of David Tour considering 5 samples	60
7.1	Instance order based on number of coordinates	76

Index

ATSP, 5

Classification, 1, 21

Data Mining, 27

David Tour, 10

Default Classification, 23

Euclidean TSP, 7

Evolutionary Computation, 22

Feature, 3, 27

Fractal TSP, 2, 7

Fratcal TSP, 1

GABIL, 25

GAssist, 2, 3, 25

Generalized TSP, 7

Genetic Algorithm, 22

Global Statistics, 3, 39

Good-enough Solution Finder, 72

Iterative Rule Learning approach, 25

Koch Tour, 10

L-System, 2, 7, 8

LCS, 1, 2, 21

Machine Learning, 21

Michigan approach, 24

MNPeano, 11

MPeano, 11

Optimisation Problem, 1

Order, 8

Partition, 49

Partition Statistics, 3, 49

Pittsburgh approach, 22

Reinforcement Learning, 24

Sample Statistics, 3, 59

Scrambled Statistics, 3, 65

STSP, 5

Triples, 3, 30

TSP, 1, 2, 5

XCS, 24

ZCS, 24