LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-215690

Graph-based Techniques for Orbit Classification: Early Results

A. Bagherjeiran, C. Kamath

September 27, 2005

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Graph-based Techniques for Orbit Classification: Early Results

Abraham Bagherjeiran
Lawrence Livermore National Laboratory
bagherjeiran2@llnl.gov

Chandrika Kamath
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
kamath2@llnl.gov

11th April 2006

Abstract

An important step in the quest for low-cost fusion power is the ability to perform and analyze experiments in prototype fusion reactors. An automated analysis and interpretation software toolkit will allow physicists to quickly analyze their experiments and plan new ones. In this preliminary report, we consider the analysis of Poincaré plots that contain the orbits generated by the particles in a fusion reactor. We describe how we can use graph-based methods to extract features from orbits. These features are then used to evaluate the accuracy of machine learning algorithms and their response on orbits with few points.

Contents

1 Introduction

An important step in the quest for low-cost fusion power is the ability to perform and analyze experiments in prototype fusion reactors. Devices such as the National Compact Stellarator Experiment (NCSX), which is under construction at the Princeton Plasma Physics Laboratory (PPPL), as well as the smaller Current Drive Experiment (CDX) tokamak, allow physicists to perform magnetic confinement experiments. These experiments determine the best shape for the hot reacting plasma and the magnetic fields necessary to hold it in place. In addition, advances in computational resources, such as parallel computers, massive data stores, and fast data transfer rates, have made it possible to simulate the experiments computationally in three dimensions over time. These simulations allow the physicists to design new reactors as well as to select the parameters to be used in experiments. The results from the experiments are, in turn, used to validate the simulations. Thus, the analysis and interpretation of the data from both simulations and experiments is a key step in the understanding and development of fusion reactors. In this report, we show how data mining techniques can be used in the analysis of data from fusion devices.

In our work, we focus on the problem of the analysis of Poincaré plots. These two-dimensional plots are obtained for planes which intersect the torus-shaped device. A plot consists of several orbits, each corresponding to a particle moving in the device. An orbit is composed of several points, each point representing the intersection of the particle with the plane as it goes around the device. Thus, an orbit describes the motion of a particle inside the device, and consists of points which indicate where the particle intersected a specific plane in its trajectory around the device. The positions of the points in an orbit describe a particular shape. Depending on the shape, we can assign a class label to an orbit, indicating if it is a separatrix, an island chain, or a quasi-periodic orbit. The physicists involved with the NCSX experiments and the corresponding simulations are mainly interested in features which describe certain orbits such as the width of the lobes of a separatrix or the location of the centers of the islands. Thus, the first task in our analysis is to infer the class label of an orbit and then extract the features of interest from it.

In the work described in this report, we consider orbit data from both simulations and experiments. In simulation data, it is possible to obtain the points which are explicitly assigned to an orbit. Further, by running the simulation longer, we can generate a large number of points for each orbit. In contrast, the plots from experimental data consist of a few points from multiple orbits, without an explicit assignment of points to orbits. Thus, in our analysis, we need to consider each orbit individually for simulation data and consider the entire plot for experimental data.

This splits the analysis of Poincaré plots into two separate problems. In the single-orbit problem, we infer the class label of individual orbits by first extracting features from a graph representation of an orbit and then applying a classification algorithm. In the multi-orbit problem, we identify regions in the plots that are likely to be of interest to the physicists by first applying classification at the point level and then using density-based clustering algorithms to identify interesting dense components in the plots.

Our initial results show some progress toward these goals. For single-orbit classification, the use of graph-based features yields accurate classifiers that are robust when the number of points is small. Further, these algorithms are easy to insert into the analysis software. For multi-orbit analysis, our early results show that our approach can identify many of the interesting components.

This report is organized as follows. Section 2 provides an overview of the problem of orbit classification and our analysis approach. Section 3 details our work in single-orbit classification and presents experimental results. Section 4 describes our approach to multiple-orbit classification and presents preliminary results. Section 5 is a summary of our approach for the analysis of single and multi-orbit plots. Section 6 presents related work in the field of orbit classification and in the area of machine learning for plasma physics. Finally, Section 7 presents the conclusions and plans for future work.

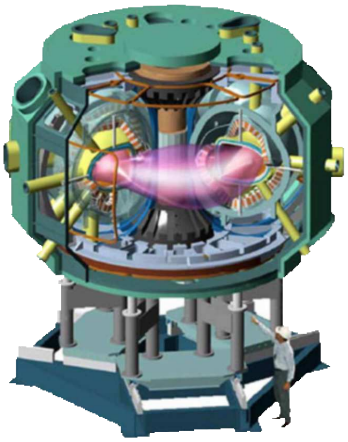


Figure 1: A schematic of the reactor for the National Compact Stellarator Experiment currently under construction at the Princeton Plasma Physics Laboratory.

2 Overview

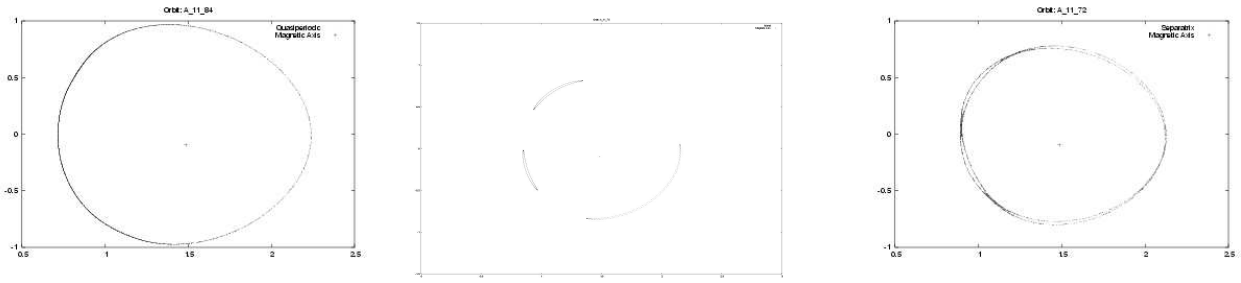
Figure 1 shows the schematic of the NCSX currently under construction at PPPL. A particle moving around the torus will trace out a three-dimensional trajectory over time. Consider a plane intersecting the torus perpendicular to the magnetic axis. Let a point in this plane be the intersection of the trajectory of the particle with the plane as it starts to move through the torus. After it completes one round through the torus, it will likely intersect the plane at a different point. The intersections of this trajectory with the plane as the particle keeps going around the torus form an orbit. It is possible for a particle to intersect the plane at a given location more than once, in which case it will trace out the orbit again (in the absence of floating point errors).

In this report, an **orbit** $R \subseteq \mathbb{R}^2$ is the set of points at which a particle intersects a plane for the duration of the experiment. Depending on the shape of the orbit, it can be assigned a class label. Three orbits, each with a different class label are depicted in Figure 2. From these examples, we see that the orbits are visually very different. Notably, the quasi-periodic orbit appears to be a thin, closed curve. The island chain orbit has three distinct islands that are thin in this particular example. Like the quasi-periodic orbit, the separatrix orbit appears closed but has radial gaps called lobes; there are four such lobes in this particular orbit. The separatrix orbit can be considered to be one where there are several islands, whose tips merge, end-to-end.

In isolation, the orbits in Figure 2 are easily distinguishable, at least visually. This is the case in simulation data where we can obtain each orbit individually. However, we must also obtain the interesting characteristics of individual orbits from experimental data, where we only have multiple orbits, without the explicit assignment of points to individual orbits. We define a **multi-orbit plot** as

$$\mathbf{R} \;=\; \bigcup_{i=1}^{n} R_i \;,$$

that is, the union of the points contained in n individual orbits. Such a multi-orbit plot is referred to in the literature as a Poincaré plot, or a "puncture plot" (see Figure 3). Essentially, it is a collection of orbits on the plane where each orbit is generated by a different particle. When considered as a whole, the multi-orbit plot can contain orbits from all classes. Classifying individual orbits and finding their characteristics appears to



(a) Quasi-periodic (b) Island Chain (c) Separatrix

Figure 2: An example of an orbit from each class from simulations of the CDX tokamak. The cross indicates the approximate magnetic center.

be more difficult in this case than in the single-orbit classification case.

To help with the analysis, we identify interesting spatial regions in each orbit which we call **components**. For example, in a separatrix orbit, the components are **cross points** and **lobes**. Cross points are regions of dense points that indicate the intersection between the separate arcs of the orbit. Figure 2 shows a separatrix orbit with 4 cross points: three on the left and one on the middle-right. Separatrix lobes are the empty interior regions between the arcs and cross points. The components for an island-chain orbit are the individual **islands** within it. The physicists are interested in the characteristics of these components such as the width of a separatrix lobe or the center of each island in an island chain.

2.1 Sample Data

In general, the data under analysis consist of a set of points. The points constitute either individual orbits from simulations or multiple orbits from experiments. For simulation data, we can also consider all orbits in a plane at the same time, leading to the multi-orbit case.

In simulation data, we receive individual orbits like those shown in Figure 2. This is ideal because one can simulate thousands of points in the orbit. The simulation data do, however, contain floating point errors that appear as noise as may be evident in the cross points of Figure 2c. As the error builds up over time, only the first 1000 or so points are used for an orbit from simulation data. In the experiments conducted in this report, we used 6 multi-orbit plots each containing 100 orbits from simulations of the CDX tokamak. The number of points in each orbit is variable. Table 1 lists the orbits in each plot and the hand-labeled true class for each orbit for future reference.

Although we have no experimental data at present, we do not expect that they will contain individual orbits. The experimental data will resemble the plot in Figure 3 without the class labels associated with each orbit. The data are also expected to contain 50 to 100 points per orbit in contrast to the more than 2000 points present in most orbits in Figure 2. At this level of point detail, many of the distinguishing features present in the classes are less pronounced. This presents a classification difficulty which we detail in Section 3.6.

2.2 Goals of the Analysis

There are several pieces of information we wish to extract from the puncture plots. First, we are interested in assigning a class label to each orbit. This is for the case where the data for each orbit is available separately. In addition to the class label, the physicists are interested in the characteristics of some of the orbits, for

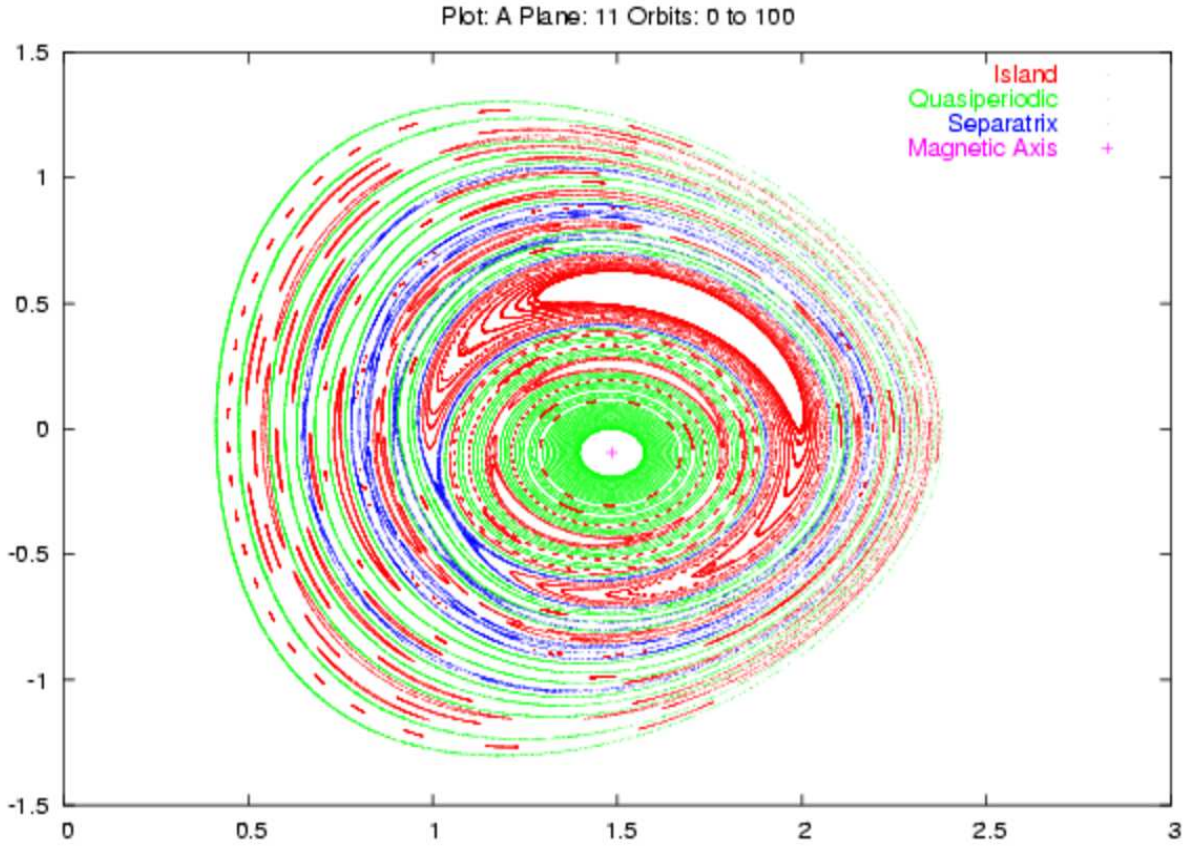


Figure 3: An example of a multi-orbit plot. This plot contains 100 orbits, including the three from Figure 2. Each orbit is color coded to indicate its class label.

#	0	#		10		11		12		13		14	
0	I	54	Q	I	I	Q	I	Q	I	Q	I	I	I
1	Q	55	I	I	I	Q	I	Q	I	Q	I	I	I
10	Q	56	Q	Q	I	Q	I	I	I	Q	I	Q	I
11	Q	57	I	Q	I	Q	I	Q	I	Q	I	I	I
12	Q	58	Q	Q	I	Q	I	Q	I	Q	I	Q	I
13	Q	59	S	Q	I	Q	I	Q	I	Q	I	Q	I
14	Q	6	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
15	Q	60	S	Q	I	Q	I	Q	I	Q	I	Q	I
16	Q	61	I	Q	I	Q	I	Q	I	Q	I	Q	I
17	Q	62	Q	Q	I	Q	I	Q	I	Q	I	Q	I
18	Q	63	I	Q	I	Q	I	Q	I	Q	I	Q	I
19	Q	64	I	Q	I	Q	I	Q	I	Q	I	Q	I
2	Q	65	Q	Q	I	Q	I	Q	I	Q	I	I	I
20	Q	66	I	Q	I	Q	I	Q	I	Q	I	Q	I
21	Q	67	I	Q	S	Q	S	Q	I	Q	I	Q	I
22	Q	68	S	Q	Q	I	Q	Q	Q	Q	I	Q	I
23	Q	69	I	Q	Q	Q	Q	Q	Q	Q	I	Q	Q
24	Q	7	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
25	Q	70	Q	Q	I	Q	I	Q	I	I	I	Q	S
26	Q	71	Q	Q	Q	I	Q	Q	Q	Q	S	Q	I
27	Q	72	S	I	S	Q	S	Q	I	Q	I	Q	I
28	Q	73	Q	Q	I	Q	Q	Q	I	Q	I	Q	Q
29	Q	74	S	I	Q	Q	I	Q	Q	I	I	Q	I
3	Q	75	S	Q	I	Q	I	Q	I	Q	I	Q	I
30	I	76	I	Q	I	I	I	Q	S	Q	Q	Q	Q
31	I	77	I	Q	Q	Q	S	Q	Q	Q	Q	Q	Q
32	Q	78	Q	Q	S	Q	S	Q	I	I	I	I	I
33	Q	79	Q	Q	I	Q	I	I	I	I	I	I	S
34	Q	8	Q	Q	Q	I	Q	I	Q	I	Q	I	Q
35	Q	80	S	Q	I	I	S	I	I	I	Q	I	Q
36	Q	81	Q	I	Q	I	Q	I	I	Q	Q	Q	I
37	Q	82	Q	S	I	I	I	Q	I	I	I	Q	I
38	I	83	S	Q	I	Q	I	Q	I	I	I	Q	I
39	Q	84	I	I	Q	Q	Q	I	Q	S	I	Q	Q
4	Q	85	Q	Q	I	Q	I	Q	I	Q	S	Q	S
40	Q	86	I	Q	Q	Q	Q	I	Q	Q	Q	I	S
41	Q	87	S	Q	I	I	I	I	I	Q	I	Q	I
42	S	88	Q	Q	S	I	S	Q	I	I	I	Q	I
43	Q	89	Q	I	Q	Q	Q	Q	Q	I	I	Q	Q
44	Q	9	I	Q	Q	Q	Q	I	I	I	Q	Q	Q
45	Q	90	I	Q	I	I	I	Q	I	Q	I	I	I
46	Q	91	Q	S	I	I	Q	I	Q	Q	I	S	Q
47	I	92	Q	Q	I	Q	I	S	I	I	I	I	I
48	Q	93	S	Q	Q	S	Q	I	Q	I	Q	I	I
49	Q	94	Q	I	I	I	I	I	I	I	S	I	S
5	Q	95	Q	Q	I	Q	I	Q	I	Q	I	Q	Q
50	I	96	Q	I	I	I	I	I	I	I	I	I	I
51	Q	97	Q	I	S	I	Q	I	I	I	Q	I	Q
52	Q	98	Q	I	I	I	I	I	I	I	Q	I	Q
53	Q	99	Q	I	Q	I	Q	I	Q	I	Q	I	Q

Table 1: Hand classification results for plots in planes 0, 10, 11, 12, 13, and 14 in file cdx_2D.3d.h5. There are 100 orbits in a plot, listed as two columns. Columns 1 and 3 provide the orbit number. Columns 2 and 4 are the labels for plane 0, columns 5 and 6 are for plane 10, columns 7 and 8 are for plane 11, *etc.* This table is included for future reference.

example, the number of islands in an island chain, the width of each island (measured radially), the location of the centers of the islands, the width of the lobes of the separatrix, etc. We are also interested in seeing how our analysis algorithms perform when the number of points is small so that some of the characteristics of an orbit are difficult to discern.

An additional observation on these plots makes the analysis even harder. Each of the 100 orbits in Figure 3 is generated by using a "starting point" at a location on the $y = 0$ line to the right of the magnetic axis represented by the cross. The location of this starting point determines the trajectory of the particle, generating orbits of different classes. One of the main characteristics of interest is the width of the lobes of the separatrices in the plot. Because only a few starting points are chosen to create the plot, it is likely that a starting point corresponding to a separatrix is not selected. As a result, the separatrix will not appear in the plot, even though it is there. So, we need to find other means of obtaining the width of its lobes.

One way of doing this is to exploit the set of orbit consistency rules which must be satisfied by neighboring orbits (see [18], page 101). As explained in [18], Section 4.2, these rules can be derived using the natural constraints occuring in the physical problem. In our particular example, we can exploit the consistency rule which states that an island chain with a quasiperiodic orbit as a neighbor, indicates the presence of a missing separatrix. For example, in Figure 3, there is a large single island orbit which appears as a crescent in the top part of the figure, roughly mid-way radially between the inner-most and outer-most orbits. This island is actually the inner-most of a series of nested islands. The outermost of these is the separatrix, whose single cross point (where the tips of the island merge) occurs approximately at $x = 1.1$, $y = -0.4$. If this separatrix were missing, an approximation to the width of its lobe would be the width of the largest of the nested islands. As a result, the width of the islands in an island-chain orbit is also important. As an extreme case, it may be that not only is the separatrix missing from the plot, but the islands associated with the separatrix are also missing because their starting points were not chosen. In this case, finding the approximate width of the separatrix would require using the fact that the first quasi-periodic orbit outside the missing islands and separatrix is deformed by their presence.

Our initial focus is in labeling each of the orbits when they are available separately. We also include some initial work on the analysis of an entire puncture plot, i.e. the multi-orbit case.

2.3 Analysis Pipeline

The orbit classification work described in this paper is one approach to automating the analysis and interpretation of orbit data from experiments and simulations of fusion reactors such as the NCSX stellarator and CDX tokamak. An example of a possible analysis pipeline is shown in Figure 4 where we use data mining algorithms in various stages of the analysis. In simulations, one can easily provide individual orbits as input to single-orbit classifiers. In experiments, however, individual orbits may not be available and we would need to use multi-orbit classifiers. At the end of the analysis process, the objective is to extract features representing each plot, including the number and location of the islands, the width of the islands, the width and location of the separatrices, etc. These features can then be catalogued and queried as appropriate. For example, scientists might be interested in retrieving all plots where no islands have width greater than a certain value. Such information can be used to understand both existing simulations and experiments and plan new ones.

The database containing the orbit and plot information could include features derived directly from the plots, or indirectly inferred from them. For example, Figure 2c shows a separatrix orbit with 4 lobes. If this separatrix were absent from a multi-plot orbit so that it only contained the islands within in the lobes, we can use the island information to conclude where the separatrix should be. We know that the islands must be smaller than the extent of the lobes and that a separatrix cross point must occur between individual islands; this allows us to localize the potential separatrix lobes. Such domain knowledge could also be encoded into the analysis system.

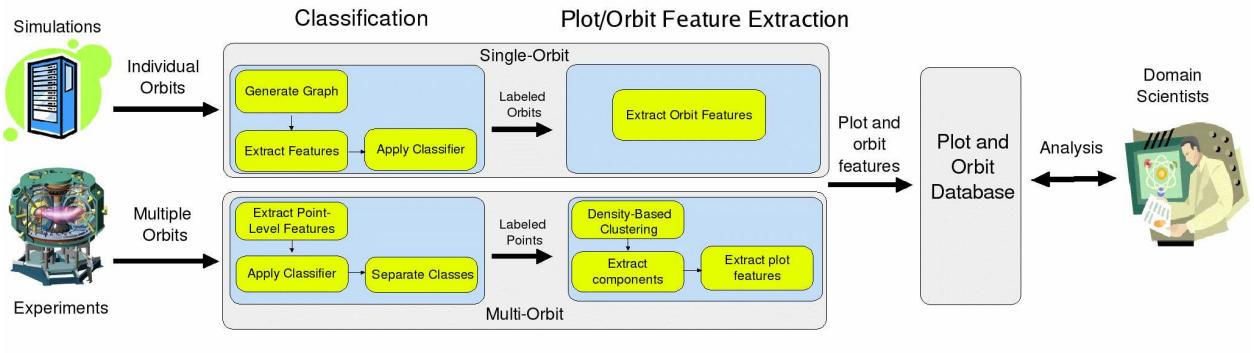


Figure 4: A block diagram showing the application of machine learning and data mining for automated analysis and interpretation of orbit data.

3 Single-Orbit Classification

In this section, we describe our work on the classification of a single orbit of a Poincaré plot. Our work focuses on a graph-based approach to orbit classification. We first discuss the preprocessing of a graph and show how we can use graph properties to extract features for several classification algorithms. We next describe an existing orbit-classification algorithm called KAM [18] and discuss how we have customized it for our data sets. The features extracted for KAM and our customized version can be used in more traditional classification algorithms such as decision trees and neural networks. Using the data described in Section 2.1, we evaluate the accuracy of the different classifiers when the number of points for each orbit is reduced.

3.1 Graph Preprocessing

We follow a graph-based approach to orbit classification based on an existing algorithm specifically designed for orbit classification [18]. The approach requires a significant amount of preprocessing that first constructs a graph representation of the points of an orbit. Using this orbit graph, we partition the graph into clusters and then use domain-level heuristics to compute several properties on the vertices and edges in the graph. These properties are the basis for features used in several classification algorithms.

3.1.1 Definitions from Graph Theory

To understand the remainder of this section, it is necessary to first review a few definitions from graph theory.

Graph A graph $G = \{V, E\}$ is a set of vertices V and edges E.

Edge An edge $e \in E$ is a pair of vertices $e = \langle v_1 \in V, v_2 \in V \rangle$.

Adjacency Two vertices $v_1 \in V$ and $v_2 \in V$ are adjacent–written $adj(v_1, v_2)$–if and only if an edge links $v_1$to v_2:

$$adj(v_1, v_2) \iff \langle v_1, v_2 \rangle \in E$$

Degree The degree $deg(v)$ of a vertex $v \in V$ is the number of adjacent vertices to v:

$$deg(v) = |\{u \mid adj(u, v)\}|$$

Vertex Properties Vertex properties are defined for each vertex in the graph and hold either Boolean or real-valued information for each vertex. For example, we use coordinates $(x(v)\ ,\ y(v))$ as properties for vertices and define the distance between vertices as the Euclidean distance between the coordinates of the vertices

$$\begin{aligned} w &= d(v_1, v_2) \\ d(v_i, v_j) &= \sqrt{(x(v_i) - x(v_j))^2 + (y(v_i) - y(v_j))^2}\ . \end{aligned}$$

Since KAM refers to vertices as nodes, we use the terms node and vertex interchangeably.

Edge Properties An edge, e, has a real-valued property $length(e) \in \mathbb{R}$ equal to the Euclidean distance between its vertices.

3.1.2 Generating the Orbit Graph

The first preprocessing step is to compute a graph representation of an orbit. First, the algorithm creates a graph without edges whose vertices are the points in the orbit. Next, it computes the Delaunay triangulation of the graph and then computes the Minimal Spanning Tree (MST) of the triangulation. The MST of a graph is a tree that contains all vertices of the graph but the sum of the edge lengths in the tree is minimal. The use of the MST as the orbit graph as opposed to some other graph representation is based on the observation by Yip that it characterizes the shape of the orbit and is similar to the way humans see objects: "Ultimately, the choice of the MST representation can be justified only by its usefulness as a basis for extracting shape information necessary for orbit [classification]" [18]. We also suspect that the MST is used because well-known algorithms exist to compute it efficiently. For example, computing the Delaunay triangulation before the MST greatly improves the computation time. Since the tree has fewer edges than the triangulation and no cycles, further processing is faster than with the triangulation. For the remainder of this paper, we will refer to the MST of the orbit as the orbit itself or the orbit graph.

3.1.3 Partitioning the Graph

The next step is to partition the graph into clusters by removing edges. The algorithm used in our implementation removes all edges whose weight exceeds a threshold $t_{partition}$. In other words, vertices which are more than $t_{partition}$ apart in Euclidean distance are removed. The partitioning process yields k independent subgraphs. Each node in a subgraph is assigned a *cluster* property which is set to the index of the subgraph to which the node belongs.

3.1.4 Computing Graph Properties

The last preprocessing step applies to the whole orbit graph and all the partitioned subgraphs. The result is an assignment of properties to the vertices and edges. These properties serve as the basis for the features that will be used for classification.

Diameter The diameter is the defined as the longest shortest path between any two vertices in the graph. Although this is a difficult problem for graphs in general, it is simple for trees. The shortest path–with respect to edge length–between any two vertices u and v in a tree is the depth-first search path from u to v; thus, the diameter is the longest depth-first path between any two vertices in the tree. To compute the diameter, the algorithm starts at an arbitrary vertex u and finds the vertex u' at the end of the longest depth-first path from u. The diameter is then the longest path emanating from u'. At most, the computation requires two maximum-length depth-first traversals of the tree. The vertices along the diameter have the Boolean property *diameter* set to *true*; otherwise, it is set to $false$. Next, the algorithm identifies the beginning and ending endpoints of the diameter for later use in feature extraction.

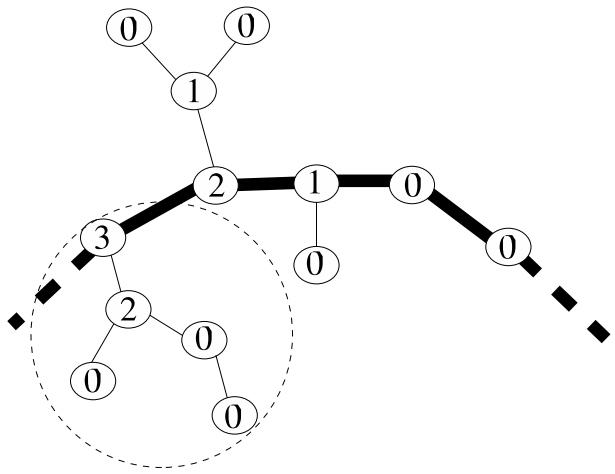


Figure 5: Branch calculation example on a small portion of a larger graph. The thick edges indicate the diameter of the graph. Vertex labels show the value for the *branchLength* property.

Branches For each vertex v, with degree 3 or more, the algorithm computes the *branchLength* property as the length of the longest path starting at v and not passing through any other vertex $u \neq v$ along the diameter. As illustrated in Figure 5, vertices with degree less than 3 are ignored and assigned $branchLength(v) = 0$. Using the *branchLength* property, the algorithm assigns two additional Boolean properties on vertex v

$$shallowBranch(v) = \begin{cases} true & \text{if } branchLength(v) \geq t_{shallow} \\ false & \text{otherwise} \end{cases}$$

$$deepBranch(v) = \begin{cases} true & \text{if } diameter(v) \wedge (branchLength(v) \geq t_{deep}) \\ false & \text{otherwise} \end{cases}$$

given thresholds $0 < t_{shallow} < t_{deep}$. The *primaryBranch* property

$$primaryBranch(v) = \begin{cases} true & \text{if } diameter(v) \wedge shallowBranch(v) \wedge \\ & (branchPath(v) = \max_w branchLength(w)) \\ false & \text{otherwise} \end{cases}$$

is assigned to the branch vertex v whose path is the longest branch path but may not necessarily be a deep branch. After assigning the *primaryBranch* property to all vertices along the diameter, the value of the property is replicated to all non-diameter descendants of a primary branch vertex.

In the example of Figure 5, all edges have length 1 and the labels of the vertices indicate the value of the *branchLength* property. With the threshold $t_{shallow} = 1$, the graph contains 5 vertices with a *true* value for *shallowBranch* property. With threshold $t_{deep} = 2$ and proceeding along the thick edges of the diameter, the graph contains 2 deep branches vertices having branch lengths 2 and 3. The second branch vertex in the graph with branch length 2 is not a deep branch vertex because it is not rooted along the diameter. The branch vertex with length 3 is assigned a *true* value for the *primaryBranch* property. Lastly, all vertices in the circled region are assigned a *true* value for the *primaryBranch* property because they are located along the primary branch.

Primary Nodes A primary node v has the following property:

$$primary(v) = \begin{cases} true & \text{if } diameter(v) \vee primaryBranch(v) \\ false & \text{otherwise} \end{cases}$$

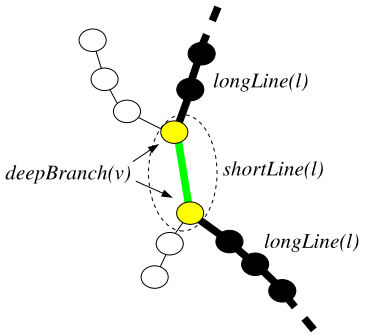


Figure 6: Example of hyperbolic structure for a portion of a separatrix orbit. The thick lines and filled circles indicate the diameter of the graph. The circled region is the cross point.

Primary nodes are the nodes along the diameter or are contained in the longest branch whose root is at the diameter. The intuition behind this property is that for some graphs, the diameter may not contain a large branch in the graph. The primary node property is designed to compensate for the large branch not on the diameter by adding the primary branch.

Hyperbolic Structure The last set of properties characterizes the branching structure of the graph. The properties are specifically designed to describe separatrix orbits. Figure 6 shows an example of the ideal cross point of a separatrix orbit. The graph contains two deep branch vertices as indicated in the figure. Following the set of edges D along the diameter, the algorithm adds each edge $e \in D$ to a set l. When it encounters a deep branch, it saves the current set $L = L \cup l$ and creates a new set $l = \emptyset$. This continues for the remainder of the diameter. At the end of the process, the edge sets L form a mutually exclusive partitioning of the diameter. Next, the algorithm assigns one of two labels to each line segment $l \in L$:

$$
\begin{aligned}
shortLine(l) &= \begin{cases} true & \text{if } length(l) < t_{hyperbolic} \\ false & \text{otherwise} \end{cases} \\
longLine(l) &= \begin{cases} true & \text{if } length(l) \geq t_{hyperbolic} \\ false & \text{otherwise} \end{cases} \\
length(l) &= \sum_{e \in l} length(e)
\end{aligned}
$$

where the length of a line segment is the sum of the lengths of its constituent edges. The result of this labeling is a sequence of labels that characterizes the diameter. In the example of Figure 6, the line segment sequence is

$$\ldots \quad longLine, deepBranch, shortLine, deepBranch, longLine \quad \ldots$$

proceeding from top to bottom along the thick edges of the diameter of the figure. The orbit has a hyperbolic structure if and only if the line segment sequence contains the pattern *longLine*, *deepBranch*, *shortBranch*, *deepBranch* at least twice. Interestingly, the pattern is not symmetric, but this is only relevant if the pattern occurs at the end of the diameter. The vertices and edges that form the pattern are called hyperbolic segments. The two deep branches in the hyperbolic segment are then assigned a *true* value for the *hyperbolicBranch* property. The cross point of the separatrix is the short line between the hyperbolic branches and is circled in the figure.

Property	Thresholds
$diameter$	
$shallowBranch$	$t_{shallow}$
$deepBranch$	t_{deep}
$primaryBranch$	
$hyperbolicBranch$	$t_{hyperbolic}$
$primary$	
$cluster$	$t_{partition}$

Table 2: Properties extracted from the orbit graph and their corresponding thresholds.

3.1.5 Summary

As shown in Table 2, we extract several vertex-, edge-, and graph-level properties from the minimal spanning tree of the points in the orbit. The properties depend on selecting good threshold values. This is a difficult problem that we address in Section 3.4. Given good thresholds, we use these properties to derive several features for use in the KAM rules and other classifiers.

3.2 Single-Orbit Classification Algorithms

The properties described in the previous section are the basis for features for the default KAM, a customized version of KAM, and several standard machine learning classifiers.

When describing the features, all lengths are given as fractions of the diameter length. Since orbits can vary in size, the reference to the diameter length makes length-based features robust to the size of the orbit. We use the following notation in the rest of this section.

Notation

f_X A feature that is a fraction with range $[0, 1]$.

n_X Number of vertices that satisfy property X. Since all vertices have the property $node(v)$, the value n_{node} is the number of vertices in the graph.

3.2.1 The Default KAM classifier

For the default KAM classifier, we first extract the features described in the book [18] and then apply a set of static rules on the features. We also address the problem of selection of thresholds used in the calculation of the features. We use the following features for the default KAM classifier:

d The Euclidean distance between the initial and final vertices of the diameter as a fraction of the total path length between the vertices.

$f_{hyperbolic}$ Fraction of hyperbolic nodes with respect to shallow branch nodes: $\frac{n_{hyperbolicBranch}}{n_{shallowBranch}}$. Separatrices should have high values for this feature; all other classes should have low values.

$f_{primary}$ Fraction of primary nodes: $\frac{n_{primary}}{n_{node}}$. Quasi-periodic orbits should have high values, islands medium, and separatrices low.

$n_{shallow}$ Number of shallow branch nodes.

$f_{shallow}$ Fraction of shallow branch nodes: $\frac{n_{shallow}}{n_{node}}$. Quasi-periodic orbits should have low values; separatrices should have high values.

Rule	Formula
QPCluster	$n_{clusters} = 1 \wedge (n_{shallow} < p_{branch}) \wedge (d \leq p_d)$
Quasiperiodic	$QPCluster \wedge Origin$
Island	$(n_{clusters} \geq 1) \wedge (f_{QP} \geq p_{QP})$
Separatrix	$Hyperbolic \wedge (f_{hyperbolic} \geq p_{hyperbolic}) \wedge (f_{primary} \geq p_{primary})$
Chaotic	$f_{primary} < p_{primary}$
Unknown	$Chaotic \vee \neg(Quasiperiodic \vee Island \vee Separatrix)$

(a)

Parameter	Value
p_d	5%
p_{branch}	1
p_{QP}	100%
$p_{hyperbolic}$	100%
$p_{primary}$	80%

(b)

Table 3: (a) KAM Rules. (b) The values of the parameters used in the rules.

$n_{clusters}$ The number of clusters found in the graph partitioning process.

f_{QP} Fraction of clusters that are quasi-periodic given the $QPCluster$ rule defined in Table 3.

Hyperbolic Whether or not the graph has a hyperbolic structure. Obviously, separatrices are the only class of orbits where this feature should be true.

Origin Whether or not the orbit contains the origin of the predefined magnetic axis. This is computed as the number of crossings of the magnetic axis along edges in the MST. There must be between 3 and 4 total crossings with at least 1 crossing in each direction. Quasi-periodic orbits should contain the origin, but islands and separatrices do not necessarily contain the origin.

After extracting the feature, the KAM classifier applies the set of static rules listed in Table 3 to determine whether an orbit belongs to one of the classes. KAM assigns to an orbit the label of the first rule it satisfies following the order of the table. We modified the rules given the constraints of our domain. Notable differences are as follows.

- We use parameters whose values can be changed instead of hard-coding the threshold values as done in KAM.
- Quasi-periodic orbits must contain the origin but not if they are clusters.
- We allow the cases of a separatrix with 1 pair of hyperbolic branches or a 1-island chain that are disallowed in the original version of KAM [18] but can appear in our dataset.
- Chaotic orbits do not appear in our dataset and are therefore labeled as unknown.

Since the KAM rules are heuristics based on a visual evaluation of orbits and domain knowledge, we wanted to compare the rules that a decision tree learns to the KAM rules. We use these features to apply the KAM classifier to all the orbits in our dataset. Next, we train a decision tree with the KAM-assigned class labels as the class labels of the data. Figure 7 shows the resulting decision tree. The decision tree learns the KAM predications to over 99% accuracy on the training set. Interestingly, the rules have some similarity to the original KAM rules.

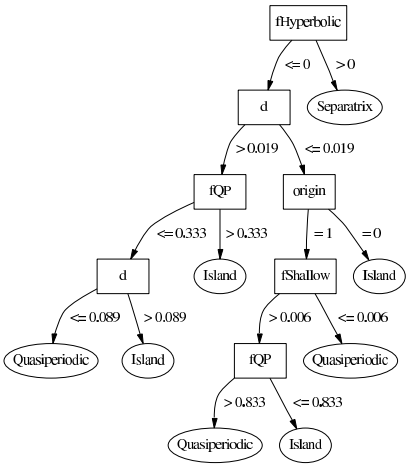


Figure 7: Decision tree trained to predict the KAM evaluation.

- Quasi-periodic orbits must contain the origin, have few branches, and have close endpoints.
- Separatrices must have a high fraction of hyperbolic branches.
- Islands must either contain quasi-periodic orbits or must be quasi-periodic orbits that do not contain the origin.

Although the default KAM classifier discussed here uses the parameters from the KAM book as shown in Table 3, the book does not recommend values for the thresholds used for the graph processing. To properly construct the features, we require values for the branch and partitioning thresholds shown in Table 2. We use a class-conditional probability distribution approach to find the thresholds.

In the class distribution approach, we separate the orbits by class and then estimate the various probability densities for each threshold. For branch thresholds, we compute and then estimate the density of the branch lengths of all branches for all orbits in each of the classes. The shallow branch threshold is the point at which the distributions of the island and quasi-periodic densities cross. The deep branch threshold is the point at which the island and separatrix densities cross. The approach thus selects the thresholds that best separate the two classes. We performed similar estimations for the hyperbolic and partition thresholds.

3.2.2 The Custom KAM classifier

Upon an analysis of the performance of the default KAM on our dataset, we realized that some changes to the original classifier could improve the results. This extension to KAM, which we call custom KAM, tries to follow the original KAM algorithm by having static rules that depend on the features from the graph. Our custom KAM classifier consists of some additions to the default KAM classifier to support some of the unique elements in our domain. For example, in addition to all the features of the default KAM classifier, we extract one additional feature and introduce one new threshold, as follows:

f_{LS} Fraction of clusters that are individually line segments. A line segment is a graph whose Euclidean distance between the initial and final vertex of the diameter is a large fraction of the total diameter path length with respect to the threshold t_{LS}.

Further, to allow more expressive rules in our custom KAM algorithm, we use real-valued instead of Boolean predicates and real-valued combination operators. The purpose is to allow for a partial evaluation (instead of a Boolean evaluation) and to make the evaluation tolerant to poorly selected thresholds and parameters. This is useful both for classification and to aid the selection of parameters and thresholds. In addition, in the custom KAM classifier, we allow both the parameters and the thresholds to be changed by the user. Recall that in the default KAM classifier, we used the parameters listed in the rules defined in the KAM book, but obtained the thresholds used in the features via a distributional approach.

The quasi-periodic rule evaluation K_Q has three components that mirror the default KAM rules:

$$\begin{aligned} K_{Q_1} &= n_{shallow} \leq p_{branch} \\ K_{Q_2} &= d < p_d \\ K_{Q_3} &= Origin \end{aligned}$$

the final evaluation is the average of the component predicates. For the $QPCluster$ rule, the final evaluation is:

$$K_{QPCluster} = \frac{K_{Q_1} + K_{Q_2}}{2}$$

For the quasi-periodic rule, the evaluation is:

$$K_Q = \frac{1}{3}(K_{Q_1} + K_{Q_2} + K_{Q_3})$$

The island chain rule evaluation

$$K_I = 1 - |p_{qp} - \max\{f_{QP}, f_{LS}\}|$$

is defined as the degree to which all clusters are either individually quasi-periodic or individually line segments. The parameter p_{qp} is a user-defined fraction that conveys the maximum evaluation of the components.

The separatrix evaluation K_S has two components that also mirror the default rules:

$$\begin{aligned} \Delta_h &= \frac{p_{hyperbolic} - f_{hyperbolic}}{p_{hyperbolic}} \\ K_{S_1} &= \begin{cases} 1 - \Delta_h & \text{if } \Delta_h > 0 \\ 1 & \text{otherwise} \end{cases} \\ \Delta_p &= \frac{p_{primary} - f_{primary}}{p_{primary}} \\ K_{S_2} &= \begin{cases} 1 - \Delta_p & \text{if } \Delta_p > 0 \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

The main difference between these components and those of the default rules is that these are real-valued evaluations that are 0 when completely false, 1 when completely true, and in between to indicate a partial satisfaction of the predicate. The final evaluation is again the average of the components.

$$K_S = \frac{K_{S_1} + K_{S_2}}{2}$$

The chaotic evaluation is:

$$K_U = \gamma(1 - K_{S_2})$$

where the discount factor $\gamma = 0.75$ is designed to prevent the rule from being activated very often.

With the use of real-valued predicates, the final evaluation of the class is slightly more complicated. We use the max fuzzy disjunction operator for the final voting strategy. The predicate with the highest value determines the class. In the event of ties, the vote follows the same order as the default KAM rules. In the event that no predicate achieves a minimum value of 0.5, the final class is unknown.

The real-valued predicates allow for greater flexibility in the ultimate evaluation. For example, an incorrectly calibrated shallow branch threshold may cause a quasi-periodic orbit to have a few small branches. In the default KAM rules, this invalidates the possibility of a quasi-periodic evaluation. In the custom rules, however, the maximum evaluation would be 0.66 making the correct evaluation possible if no other predicate has a higher evaluation.

3.2.3 The Standard Machine Learning Classifiers

We use the features extracted for both the default and custom KAM classifiers to train standard machine learning classifiers. In addition to the KAM features, we include the custom KAM predicate evaluations and features that convey the underlying statistics of the edges and clusters of the orbits. As we will demonstrate in Section 3.3, these statistical features appear to be robust to few points.

μ_{length} Average edge length.

σ_{length} Standard deviation of the edge length.

u Diameter uniformity defined as the difference between the average edge length and what would occur if the points were equally spread across the diameter: $\frac{|\mu_{length} - \bar{u}|}{\bar{u}}$. If the points were distributed at equal intervals along the diameter, the average edge length would be $\bar{u} = \frac{1}{n_{diameter}}$. Quasi-periodic orbits have high diameter uniformity, island chains medium, and separatrices low.

μ_d Average diameter endpoint distance for each cluster.

$f_{cluster}$ Fraction of clusters to nodes: $\frac{n_{clusters}}{n_{node}}$ or the percent of nodes that are clusters.

$f_{cluster} - f_{shallow}$ Difference between the fraction of clusters and the fraction of shallow branch nodes. A low value for this feature means that the graph has as many branches as clusters. In this case, the orbit could be a separatrix wrongly labeled as an island chain.

K_Q Quasi-periodic predicate evaluation from the custom KAM classifier.

K_I Island predicate evaluation from the custom KAM classifier.

K_S Separatrix predicate evaluation from the custom KAM classifier.

K_U Unknown or chaotic predicate evaluation from the custom KAM classifier.

We store these features in a feature vector which permits the easy application of more traditional pattern classification algorithms. These include the standard Naïve Bayes, Multilayer Perceptron, Support Vector Machine, 1-Nearest-Neighbor, 3-Nearest-Neighbor, and Decision Tree classifiers available as part of the Weka machine learning toolkit [8, 16].

3.3 Feature Evaluation

In this section, we evaluate the discriminative ability of all the previously described features with respect to the classes by computing the information gain (*Pattern Classification* Equation 5, Page 400 [8]) with respect to the true orbit classes for each individual feature. As we are interested in orbits with a few points, we consider two cases - one where each orbit is described by 1000 points and the other where each orbit

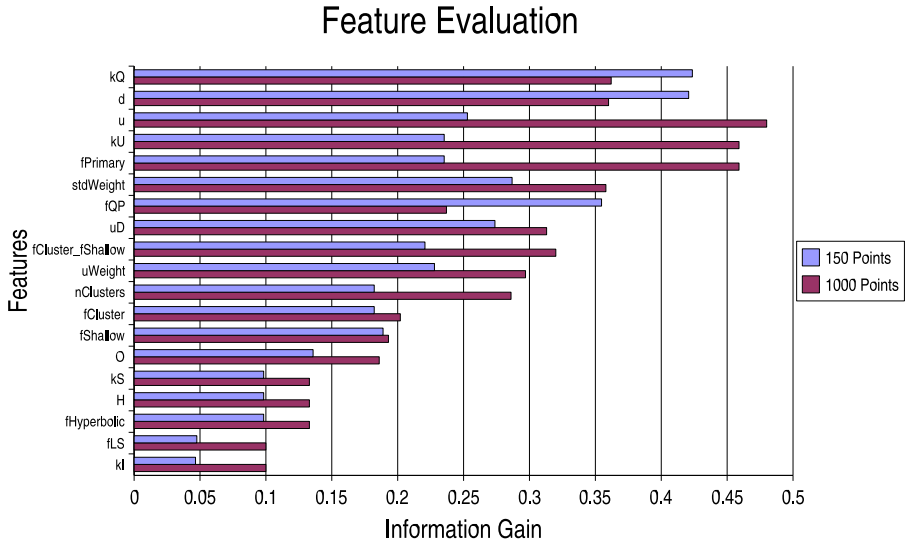


Figure 8: Comparative feature evaluation results with 150 and 1000 points. Higher values indicate better discriminative ability.

is described by the first 150 points. Figure 8 shows that most of the features have positive information gain, which indicates that each feature can individually discriminate among the classes. The most relevant features are those related to edge statistics as indicated by high values for the diameter distance d, the diameter uniformity u, and the standard deviation of the edge lengths σ_{length}. Note that σ_{length} and μ_{length} are referred to as stdWeight and uWeight in Figure 8, respectively. This suggests that the point-level density of the points is a good discriminant for the class. We expand this idea further in Section 4. Also, the chaotic evaluation also seems to be useful in that it indicates when many points lie along the diameter and is conceptually similar to the $f_{primary}$ feature.

3.4 The Selection of Parameters and Thresholds Using Genetic Algorithms

In the default KAM classifier, we fixed the parameters to the values defined in the KAM book and obtained the thresholds for the features by using a class distributional approach (see Section 3.2.1). However, in the case of the custom KAM classifier, we allow both the parameters and the thresholds to be changed. In this case, it is difficult to use a distributional approach as the parameters and thresholds are highly inter-dependent. For example, the *Hyperbolic* feature depends on the hyperbolic line length threshold $t_{hyperbolic}$ and on the branch length thresholds $t_{shallow}$ and t_{deep}. It does not explicitly depend on the partition threshold $t_{partition}$ which determines the clusters, except that the custom KAM rules would classify an orbit with many clusters as an island if the separatrix evaluation is not strong enough. We believe that a genetic algorithm can effectively search this multi-modal parameter space and therefore use it to find good thresholds for the features and parameters for the custom KAM classifier.

The genetic algorithm manipulates the thresholds from Table 2, the parameters for the default KAM classifier in Table 3, and the t_{LS} threshold for the custom KAM classifier. We used a combination of an elitist and a deterministic 3-way tournament selection strategy with a combination of high-mutation and low-mutation evolutionary operators and a constant 80% crossover probability. The high-mutation algorithm uses an Evolutionary Strategies approach that includes a mutation parameter for each index in the individual

	$n_c.$	d	$f_s.$	H.	$f_h.$	f_{pri}	$\mu_w.$	$\sigma_w.$	O.	u	μ_d	f_{QP}	f_{LS}	K_Q	K_I	K_S	K_U
Q	1	0	0	0	0	1	$\bar{u}$	0	1	0	0	1	0	1	0	0	0
I	$\star$	$\star$	$\star$	0	0	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	0	1	0	0
S	1	0	4%	1	4%	60%	$\star$	$\star$	1	$\star$	0	0	0	0	0	1	0

$$F^\star \quad = \quad \{F_Q^\star, F_I^\star, F_S^\star\}$$

Table 4: Prototypical features for each orbit class. Row 1 is the feature name, and rows 2 through 4 represent the features for quasiperiodic (Q), island (I), and separatrix (S) orbits. The $\star$ in the table is a wild-card character that is ignored by the fitness function. Features not shown in the table are absent in the fitness function.

vectors. It can find extremely fit individuals, but does not converge rapidly. The low-mutation algorithm uses the standard genetic algorithm with mutation probability of 0.1. It cannot always find the same fit individuals but frequently finds minor improvements to the current best individual. We run the two algorithms separately and manually restart them with the best individuals from both as seeds to the population. This effectively combines both a high-mutation exploratory approach with a low-mutation exploitative approach.

To evaluate an individual, the algorithm computes a fitness function based on the classification results and feature vectors using the individual on a validation set R_V of 25 orbits each with 500 points. The orbits chosen represent a mix of the possible orbits that arise in the dataset. For the evaluation, we construct the artificial feature vectors shown in Table 4 based on the feature values that represent ideal cases for each of the three classes. For example, separatrix orbits should be hyperbolic and have a high KAM separatrix evaluation. Similarly, quasi-periodic orbits should not contain any branches, have a uniform diameter, and only have a high quasi-periodic evaluation.

For each individual h, the fitness function constructs one custom KAM classifier with the parameters for each validation orbit. Next, each classifier computes a class label $K_h(R_{v_i})$ and features $F_h(R_{v_i})$. The fitness function which must be minimized is a weighted average of two factors derived from the classification:

$$\begin{aligned}
fitness(h, R_v) &= (0.6)error(h, R_v) + (0.4)r(h, R_v) \\
error(h, R_v) &= \frac{1}{25}\sum_{i=1}^{25} \delta(h, R_{v_i}) \\
\delta(h, R) &= \begin{cases} 1 & \text{if } K_h(R_{v_i}) \neq class(R_{v_i}) \\ 0 & \text{otherwise} \end{cases} \\
r(h, R_v) &= \frac{1}{25}\sum_{i=1}^{25} d(F_h(R_{v_i}), F^\star_{class(R_{v_i})})
\end{aligned}$$

The *error* fitness component is the average error of the classification with respect to the true class of the validation orbit, $class(R_{v_i})$. The rule-level fitness component r is the average Euclidean distance–d–between the features from the KAM classifier on the validation orbit R_{v_i} and the ideal feature vector based on the true class of the orbit. The weights emphasize that accuracy is more important than good features. The two components roughly convey the same information because both evaluations will be zero at perfect accuracy, but they are needed here because a genetic algorithm needs granular fitness information to discriminate between the individuals in the population. Without it, the GA cannot distinguish between two individuals that have the same accuracy but one is closer to the ideal case and could potentially lead to better accuracy.

3.5 Experiments: Single Orbit Classifiers

In our work on single-orbit classification, we conduct three experiments to evaluate the accuracy of classifiers on the dataset and their performance when the number of points in each orbit decreases. In the first experiment, we compare the two KAM implementations to standard feature-based classifiers from the Weka software toolkit [16]. In the second experiment, we evaluate the robustness of the classifiers when they are applied to orbits with few points. The desire for high accuracy on orbits with few points arises from the observation that if experimental data were to have individual orbits, they would only contain around 100 points each. Also, processing orbits with fewer points is usually faster than processing orbits with a large number of points. Finally, the third experiment tests the classifiers on previously unseen test data.

Although we tested several classifiers in our experiments, we observe that we are constrained in our ultimate choice of a classifier. Since classification is only a small part of the analysis pipeline, we require algorithms that are easy to incorporate into the analysis software. This is easily achieved with the KAM classifier because it is a rule-based classifier and simple to implement. For other algorithms that we are using from Weka, we would like to train the classifier offline and encode the results in a simple module that is easy to implement. For example, the models from the Support Vector Machine, and, to a lesser extent, the Multilayer Perceptron and Naive Bayes algorithms, are difficult to hard-code into a program. Others, such as the decision tree and even the nearest-neighbor algorithms, are comparatively simpler for a programmer to code into a program and easily update when new training data arrive. An alternative would be to use existing algorithms directly in the code by implementing them as part of the analysis pipeline.

3.5.1 Experimental Procedure

We follow a similar procedure for each of the three experiments. We tested the default and customized KAM implementations as well as several classifiers from the Weka software toolkit. For an instance-based classifier such as the nearest-neighbor algorithm, we normalized the features to be in the range $[0, 1]$.

Experiment 1: Accuracy In this experiment, we perform a comparison of the accuracy of various classifiers by performing 10 runs of 10-fold stratified cross-validation. Using our training set of 600 orbits, the classifiers both trained and tested on features extracted from orbits with N points. The number of points for each run is in the set

$$N \in \{50, 100, 200, \ldots, 1000, 1500, 1800, 2000, 2500\}$$

where the maximum number of possible points for any orbit in the dataset was 3000. The points used are the first N points in each orbit as they are known to a higher accuracy than the later points.

Experiment 2: Points In this experiment, we perform a similar cross-validation comparison by performing 10 runs of 10-fold stratified cross-validation. Instead of training and testing on orbits with N points, the classifiers trained on features from orbits with 2500 points and tested on orbits with N points. This experiment allows us to determine how well the classifiers perform when we train on orbits with a large number of points and test on orbits with a reduced number of points. The classifiers used the same orbits in the same order as in the previous comparison; only the number of points was changed.

Experiment 3: Validation In the validation experiment, we evaluate the accuracy of the classifiers on previously unseen data consisting of 100 orbits. These are difficult to classify cases composed of mainly islands and separatrices. We perform 10 runs of a modified cross-validation comparison. In each of the 10 folds of cross-validation, the classifier trains on a 90% subsample of the training data. The classifier then tests on the entire validation set. The validation data is never present in the training set and was not at all seen during the development and preliminary testing of the classifiers. The difference in testing accuracy across the runs is a result of the different training data rather than different testing data.

Dataset	CKAM	DKAM	NB	MLP	SVM	1NN	3NN	DT
50	39.67	51.67 ○	75.00±0.26 ○	81.90±0.87 ○	80.87±0.32 ○	76.98±0.83 ○	79.43±0.64 ○	81.63±0.65 ○
100	50.00	53.17 ○	78.12±0.35 ○	81.45±0.98 ○	81.10±0.12 ○	76.67±0.75 ○	78.35±0.25 ○	82.70±0.55 ○
200	68.67	58.17 ●	74.10±0.57 ○	80.88±0.38 ○	81.85±0.05 ○	76.58±0.69 ○	79.97±0.55 ○	81.32±0.51 ○
300	70.17	61.33 ●	73.30±0.13 ○	82.27±0.47 ○	81.67±0.19 ○	77.42±0.46 ○	81.15±0.81 ○	82.95±0.66 ○
400	73.67	63.17 ●	74.07±0.47	82.42±0.53 ○	82.85±0.20 ○	76.48±0.63 ○	80.00±0.36 ○	81.97±0.41 ○
500	73.00	63.00 ●	73.88±0.27 ○	83.83±0.53 ○	82.75±0.09 ○	80.50±0.54 ○	81.68±0.56 ○	82.92±0.93 ○
600	73.83	63.17 ●	74.18±0.33 ○	83.12±0.47 ○	83.35±0.15 ○	79.83±0.66 ○	82.07±0.61 ○	83.08±0.47 ○
700	73.33	63.67 ●	74.17±0.27 ○	84.27±0.57 ○	83.25±0.24 ○	81.87±0.71 ○	83.62±0.77 ○	83.90±0.64 ○
800	74.17	63.33 ●	74.30±0.28	84.73±0.56 ○	83.85±0.18 ○	83.87±0.57 ○	85.12±0.44 ○	85.48±0.81 ○
900	74.33	63.17 ●	74.53±0.28	85.63±0.68 ○	84.02±0.18 ○	84.67±0.49 ○	85.93±0.58 ○	86.13±0.56 ○
1000	74.50	63.33 ●	75.38±0.21 ○	85.75±0.71 ○	84.10±0.16 ○	84.98±0.54 ○	86.77±0.34 ○	85.73±0.59 ○
1500	73.17	61.33 ●	76.32±0.18 ○	86.90±0.74 ○	85.77±0.27 ○	85.53±0.29 ○	86.95±0.54 ○	87.40±0.85 ○
1800	75.50	60.83 ●	75.60±0.20	86.32±0.55 ○	86.75±0.37 ○	86.05±0.49 ○	87.13±0.47 ○	88.27±0.70 ○
2000	74.17	61.00 ●	76.23±0.12 ○	86.50±0.53 ○	87.02±0.34 ○	86.63±0.37 ○	88.32±0.34 ○	88.12±0.54 ○
2500	74.33	60.33 ●	75.85±0.31 ○	88.48±0.52 ○	87.57±0.57 ○	88.72±0.35 ○	90.58±0.40 ○	89.58±0.51 ○

○, ● statistically significant improvement or degradation

Dataset	DT	CKAM	DKAM	NB	MLP	SVM	1NN	3NN
50	81.63±0.65	39.67 ●	51.67 ●	75.00±0.26 ●	81.90±0.87	80.87±0.32	76.98±0.83 ●	79.43±0.64 ●
100	82.70±0.55	50.00 ●	53.17 ●	78.12±0.35 ●	81.45±0.98	81.10±0.12 ●	76.67±0.75 ●	78.35±0.25 ●
200	81.32±0.51	68.67 ●	58.17 ●	74.10±0.57 ●	80.88±0.38	81.85±0.05	76.58±0.69 ●	79.97±0.55 ●
300	82.95±0.66	70.17 ●	61.33 ●	73.30±0.13 ●	82.27±0.47	81.67±0.19 ●	77.42±0.46 ●	81.15±0.81 ●
400	81.97±0.41	73.67 ●	63.17 ●	74.07±0.47 ●	82.42±0.53	82.85±0.20 ○	76.48±0.63 ●	80.00±0.36 ●
500	82.92±0.93	73.00 ●	63.00 ●	73.88±0.27 ●	83.83±0.53	82.75±0.09	80.50±0.54 ●	81.68±0.56
600	83.08±0.47	73.83 ●	63.17 ●	74.18±0.33 ●	83.12±0.47	83.35±0.15	79.83±0.66 ●	82.07±0.61 ●
700	83.90±0.64	73.33 ●	63.67 ●	74.17±0.27 ●	84.27±0.57	83.25±0.24	81.87±0.71 ●	83.62±0.77
800	85.48±0.81	74.17 ●	63.33 ●	74.30±0.28 ●	84.73±0.56	83.85±0.18 ●	83.87±0.57 ●	85.12±0.44
900	86.13±0.56	74.33 ●	63.17 ●	74.53±0.28 ●	85.63±0.68	84.02±0.18 ●	84.67±0.49 ●	85.93±0.58
1000	85.73±0.59	74.50 ●	63.33 ●	75.38±0.21 ●	85.75±0.71	84.10±0.16 ●	84.98±0.54 ●	86.77±0.34 ○
1500	87.40±0.85	73.17 ●	61.33 ●	76.32±0.18 ●	86.90±0.74	85.77±0.27 ●	85.53±0.29 ●	86.95±0.54
1800	88.27±0.70	75.50 ●	60.83 ●	75.60±0.20 ●	86.32±0.55 ●	86.75±0.37 ●	86.05±0.49 ●	87.13±0.47 ●
2000	88.12±0.54	74.17 ●	61.00 ●	76.23±0.12 ●	86.50±0.53 ●	87.02±0.34 ●	86.63±0.37 ●	88.32±0.34
2500	89.58±0.51	74.33 ●	60.33 ●	75.85±0.31 ●	88.48±0.52 ●	87.57±0.57 ●	88.72±0.35 ●	90.58±0.40 ○

○, ● statistically significant improvement or degradation

Table 5: Results from Experiment 1 showing accuracy and significance with respect to the custom KAM (top) and decision tree (bottom) classifiers. All classifiers were trained and tested on orbits with an increasing number of points as indicated in the Dataset column.

3.6 Results

Tables 5, 6, and 7 show the results from the three experiments. We also indicate how well each classifier performs relative to the custom KAM and the decision tree classifiers. We chose the custom KAM classifier instead of the default KAM classifier as the former was designed to be more suitable for our data sets. We also chose the decision tree classifier as it is easy to extract rules from it; these rules can then be hard-coded into the analysis pipeline. Figures 9 and 10 are a graphic illustration of the results in Tables 5 and 6, and are included for clarity. When the standard deviations for the KAM classifiers are not shown, it means that they are zero because KAM deterministically applies static rules and is unaffected by training. This also explains why the results from the default and custom KAM classifiers is identical in Tables 5 and 6.

Figure 11 shows the time to extract features for an average orbit as we increase the number of points. This has been included to indicate the tradeoffs we need to make between the higher accuracy which results as we include more points in the orbit and the corresponding increase in the cost of extraction of the features.

The results for the cross-validation test (Experiment 1) show that the customized KAM classifier is a vast improvement over the original KAM in most cases. In addition, the features allowed the other traditional classification algorithms to outperform both the KAM classifiers in most cases. Interestingly, the simplest classifiers–decision tree and nearest neighbor–appear to be the overall winners in this comparison.

The results for the point test (Experiment 2), however, show that training on a large number of points and then testing on a small number of points does not seem to significantly improve accuracy results. When we compare the results of the two experiments, we observe that the performance is similar for large numbers of points. In other words, training and testing on a large number of points (in Experiment 1) gives

Dataset	CKAM	DKAM	NB	MLP	SVM	1NN	3NN	DT
50	39.67	51.67 ∘	56.73±0.80 ∘	59.97±1.18 ∘	64.27±0.29 ∘	58.12±0.80 ∘	67.98±0.51 ∘	56.05±2.62 ∘
100	50.00	53.17 ∘	69.05±0.29 ∘	68.92±0.85 ∘	73.32±0.60 ∘	69.82±0.59 ∘	76.43±0.40 ∘	67.10±1.45 ∘
200	68.67	58.17 •	72.15±0.36 ∘	72.83±0.69 ∘	76.63±0.68 ∘	78.30±1.67 ∘	79.38±0.86 ∘	69.42±1.13
300	70.17	61.33 •	73.33±0.19 ∘	75.27±0.67 ∘	78.50±0.77 ∘	80.17±0.81 ∘	80.13±0.69 ∘	70.35±0.87
400	73.67	63.17 •	73.40±0.22 •	77.42±0.78 ∘	80.15±0.76 ∘	79.62±0.96 ∘	79.65±0.86 ∘	70.92±0.94 •
500	73.00	63.00 •	75.17±0.31 ∘	77.62±0.91 ∘	80.75±0.83 ∘	77.38±0.51 ∘	77.93±0.38 ∘	71.40±1.09 •
600	73.83	63.17 •	74.68±0.32 ∘	79.38±0.62 ∘	81.95±0.65 ∘	76.35±0.54 ∘	78.68±0.35 ∘	71.52±1.26 •
700	73.33	63.67 •	75.55±0.25 ∘	80.77±0.48 ∘	82.73±0.57 ∘	74.22±0.51 ∘	77.73±0.75 ∘	71.68±1.13 •
800	74.17	63.33 •	75.47±0.19 ∘	81.10±0.40 ∘	83.17±0.48 ∘	75.95±0.34 ∘	76.35±0.39 ∘	72.62±1.05 •
900	74.33	63.17 •	75.80±0.19 ∘	82.02±0.36 ∘	83.35±0.48 ∘	75.85±0.39 ∘	77.65±0.36 ∘	78.30±0.74 ∘
1000	74.50	63.33 •	76.15±0.18 ∘	82.48±0.44 ∘	83.42±0.36 ∘	75.90±0.47 ∘	78.32±0.49 ∘	79.15±0.95 ∘
1500	73.17	61.33 •	76.58±0.24 ∘	84.45±0.44 ∘	85.43±0.34 ∘	79.12±0.53 ∘	83.55±0.44 ∘	82.88±0.75 ∘
1800	75.50	60.83 •	77.17±0.32 ∘	87.12±0.43 ∘	86.58±0.39 ∘	84.68±0.31 ∘	86.82±0.42 ∘	85.48±0.57 ∘
2000	74.17	61.00 •	77.30±0.29 ∘	87.37±0.49 ∘	86.82±0.39 ∘	87.30±0.31 ∘	88.73±0.33 ∘	85.97±0.26 ∘
2500	74.33	60.33 •	75.85±0.31 ∘	88.48±0.52 ∘	87.57±0.57 ∘	88.72±0.35 ∘	90.58±0.40 ∘	89.58±0.51 ∘

∘, • statistically significant improvement or degradation

Dataset	DT	CKAM	DKAM	NB	MLP	SVM	1NN	3NN
50	56.05±2.62	39.67 •	51.67 •	56.73±0.80	59.97±1.18 ∘	64.27±0.29 ∘	58.12±0.80	67.98±0.51 ∘
100	67.10±1.45	50.00 •	53.17 •	69.05±0.29 ∘	68.92±0.85 ∘	73.32±0.60 ∘	69.82±0.59 ∘	76.43±0.40 ∘
200	69.42±1.13	68.67	58.17 •	72.15±0.36 ∘	72.83±0.69 ∘	76.63±0.68 ∘	78.30±1.67 ∘	79.38±0.86 ∘
300	70.35±0.87	70.17	61.33 •	73.33±0.19 ∘	75.27±0.67 ∘	78.50±0.77 ∘	80.17±0.81 ∘	80.13±0.69 ∘
400	70.92±0.94	73.67 ∘	63.17 •	73.40±0.22 ∘	77.42±0.78 ∘	80.15±0.76 ∘	79.62±0.96 ∘	79.65±0.86 ∘
500	71.40±1.09	73.00 ∘	63.00 •	75.17±0.31 ∘	77.62±0.91 ∘	80.75±0.83 ∘	77.38±0.51 ∘	77.93±0.38 ∘
600	71.52±1.26	73.83 ∘	63.17 •	74.68±0.32 ∘	79.38±0.62 ∘	81.95±0.65 ∘	76.35±0.54 ∘	78.68±0.35 ∘
700	71.68±1.13	73.33 ∘	63.67 •	75.55±0.25 ∘	80.77±0.48 ∘	82.73±0.57 ∘	74.22±0.51 ∘	77.73±0.75 ∘
800	72.62±1.05	74.17 ∘	63.33 •	75.47±0.19 ∘	81.10±0.40 ∘	83.17±0.48 ∘	75.95±0.34 ∘	76.35±0.39 ∘
900	78.30±0.74	74.33 •	63.17 •	75.80±0.19 •	82.02±0.36 ∘	83.35±0.48 ∘	75.85±0.39 •	77.65±0.36
1000	79.15±0.95	74.50 •	63.33 •	76.15±0.18 •	82.48±0.44 ∘	83.42±0.36 ∘	75.90±0.47 •	78.32±0.49
1500	82.88±0.75	73.17 •	61.33 •	76.58±0.24 •	84.45±0.44 ∘	85.43±0.34 ∘	79.12±0.53 •	83.55±0.44
1800	85.48±0.57	75.50 •	60.83 •	77.17±0.32 •	87.12±0.43 ∘	86.58±0.39 ∘	84.68±0.31 •	86.82±0.42 ∘
2000	85.97±0.26	74.17 •	61.00 •	77.30±0.29 •	87.37±0.49 ∘	86.82±0.39 ∘	87.30±0.31 ∘	88.73±0.33 ∘
2500	89.58±0.51	74.33 •	60.33 •	75.85±0.31 •	88.48±0.52 •	87.57±0.57 •	88.72±0.35 •	90.58±0.40 ∘

∘, • statistically significant improvement or degradation

Table 6: Results from Experiment 2 showing accuracy and significance with respect to the custom KAM classifier (top) and decision tree (bottom). All classifiers trained on orbits with 2500 points and tested on orbits with an increasing number of points as indicated in the Dataset column.

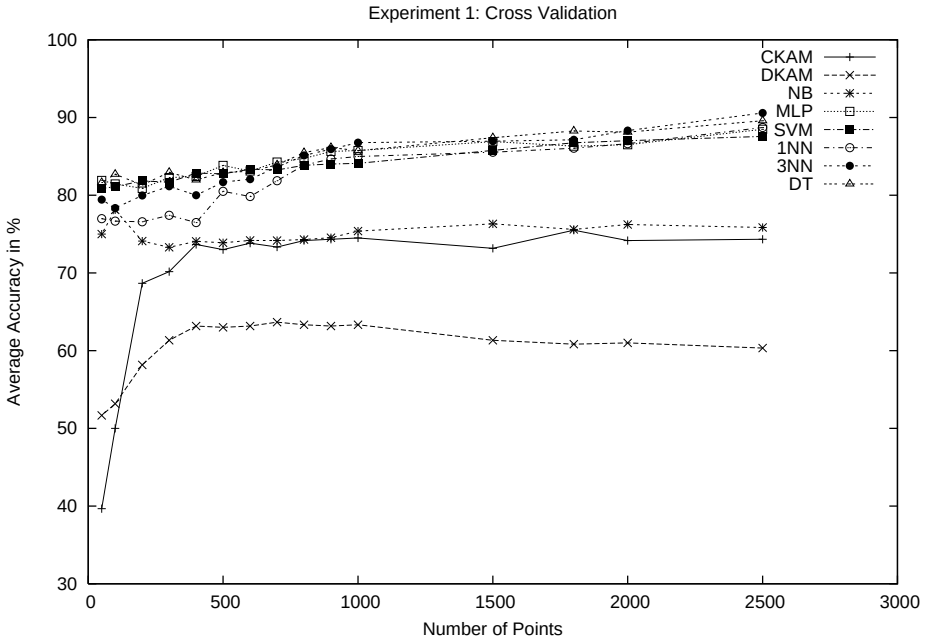


Figure 9: Comparative performance for Experiment 1.

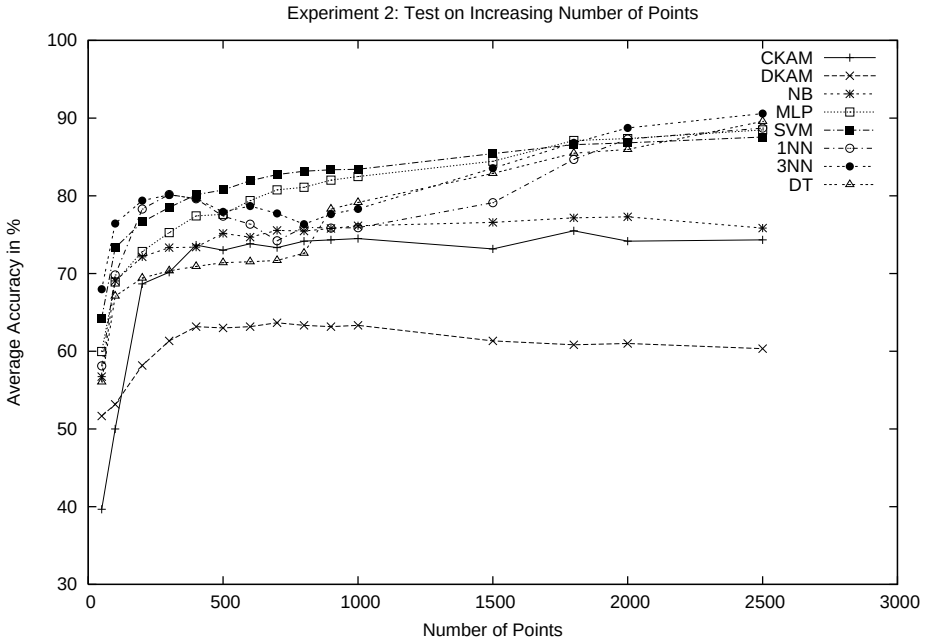


Figure 10: Comparative performance for Experiment 2.

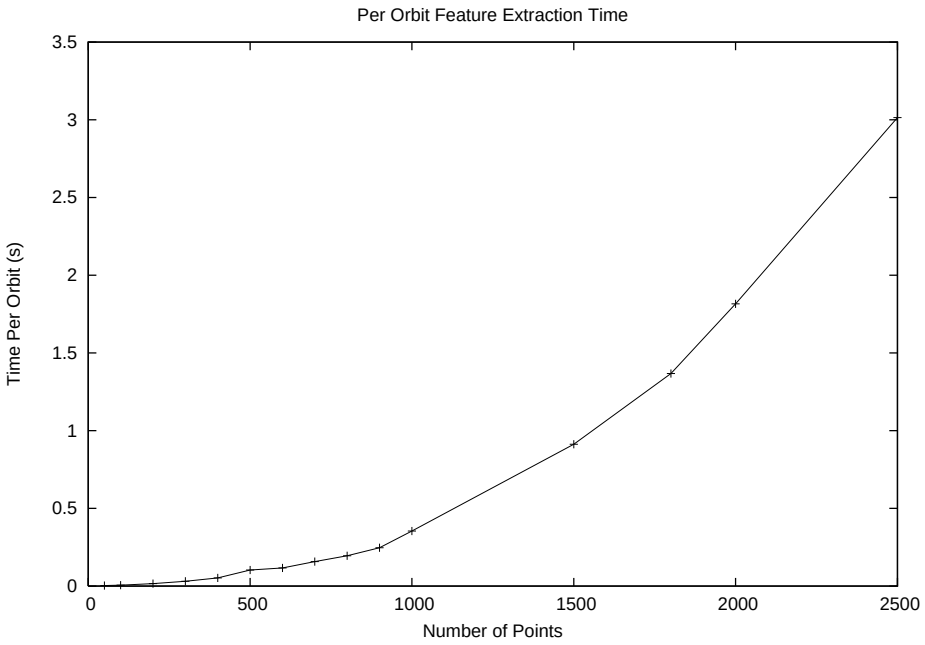


Figure 11: Per-orbit feature extraction time for orbits with an increasing number of points.

similar accuracy to training on a large number of points, but testing on a slightly smaller number of points (Experiment 2). However, for orbits with a smaller number of points (50 to 400), the accuracy is significantly improved when training on these low-point orbits. That is, training and testing on a small number of points (in Experiment 1) gives better accuracy than training on a large number of points, but testing on a small number of points (Experiment 2).

This effect of a reduced number of points in an orbit is best explained in Figure 12. The figure shows 3 example orbits with 150 and 1000 points. At 1000 points, the MST correctly identifies the outline of the shapes. At 150 points, however, the separatrix contains far too few branches because the MST forms a "zig-zag" pattern connecting points that are branches at 1000 points. In the quasi-periodic orbit, KAM detects multiple clusters misclassifying both it and the separatrix as island chains. At 1000 points, only the island chain orbit has several clusters. Although the feature-based classifiers perform better than KAM, the features used are the ones generated by the graph-based approach of KAM. Therefore, if KAM cannot correctly identify branching structure and clusters, and as a result, generates poor quality features, the feature-based classifiers will be doomed from the start.

This reliance on the KAM features also explains why the feature-based algorithms perform better when trained and tested on the same number of points. Although KAM cannot correctly identify the important components of orbits with 150 points, the classifiers only train on orbits with 150 points. Thus, they have no reason to be confused by the fact that with 150 points, the orbit is very different than it is with 1000 points. Conversely, when trained on 1000 points, the classifiers would expect orbits with few points to be very similar to those on which they have trained. Since KAM clearly does not return the same results for orbits with 150 points, the classifier suffer poor performance.

The fact that the performance is as good as it is implies that the features are only partially dependent on the number of points. In the example of the quasi-periodic orbit from the figure, the MST is visually identical with 150 or 1000 points. Although the difference is visually clear for separatrices, some features are

Dataset	CKAM	DKAM	NB	MLP	SVM	1NN	3NN	DT
100	46.88	30.21 ●	67.45±0.25 ○	67.29±0.52 ○	67.70±0.03 ○	64.66±0.89 ○	63.51±0.39 ○	66.31±0.21 ○
500	62.50	47.92 ●	72.15±0.13 ○	75.30±0.61 ○	70.02±0.20 ○	65.83±0.12 ○	69.02±0.36 ○	70.79±0.59 ○
1000	60.42	53.13 ●	71.13±0.19 ○	75.67±0.84 ○	74.35±0.28 ○	72.91±0.34 ○	69.78±0.37 ○	65.17±1.00 ○
2000	40.54	35.14 ●	75.73±0.11 ○	86.59±0.91 ○	85.84±0.26 ○	88.62±0.15 ○	86.14±0.53 ○	86.08±0.89 ○

○, ● statistically significant improvement or degradation

Dataset	DT	CKAM	DKAM	NB	MLP	SVM	1NN	3NN
100	66.31±0.21	46.88 ●	30.21 ●	67.45±0.25 ○	67.29±0.52 ○	67.70±0.03 ○	64.66±0.89 ●	63.51±0.39 ●
500	70.79±0.59	62.50 ●	47.92 ●	72.15±0.13 ○	75.30±0.61 ○	70.02±0.20	65.83±0.12 ●	69.02±0.36 ●
1000	65.17±1.00	60.42 ●	53.13 ●	71.13±0.19 ○	75.67±0.84 ○	74.35±0.28 ○	72.91±0.34 ○	69.78±0.37 ○
2000	86.08±0.89	40.54 ●	35.14 ●	75.73±0.11 ●	86.59±0.91	85.84±0.26	88.62±0.15 ○	86.14±0.53

○, ● statistically significant improvement or degradation

Table 7: Results from experiment 3 showing accuracy and significance with respect to the custom KAM classifier (top) and decision tree classifier (bottom). All classifiers trained on orbits with the number of points indicated in the Dataset column and testing on a previously unseen validation set.

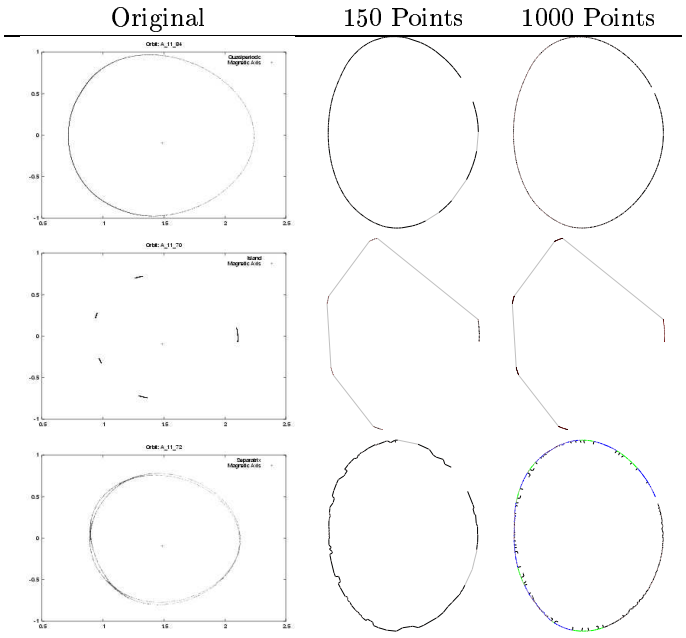


Figure 12: Orbits from Figure 2 with 150 and 1000 points.

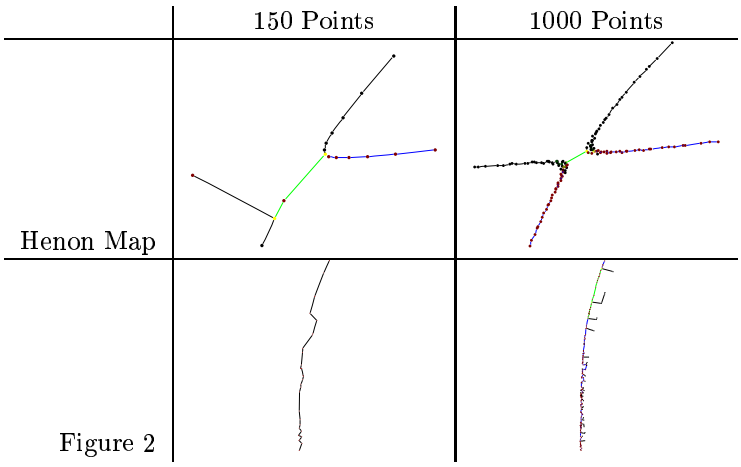


Figure 13: Zoomed-in areas of a Separatrix lobe from orbits with 50 and 1000 points. Top row: Henon map with parameters $x = 0.5551$, $y = 0.1774$, and $\alpha = 1.3284305$. Bottom row: The orbit 72 from Figure 2. Cross points are at the centers of the figures.

still visible. The similarity of quasi-periodic orbits shows that KAM can reliably detect these orbits. This is further supported from the relation of the top 3 features in Figure 8 to quasi-periodic orbits.

In addition, the accuracy of the KAM classifier worsens with more points. In contrast, the other classifiers and the custom KAM improve with more points This is largely the result of applying static rules and static thresholds under some implicit assumptions about the orbits that do not hold in our dataset.

The fundamental assumption is that the points nicely conform to the structure of the minimal spanning tree. The justification for this assumption is that points in the orbit are uniformly spaced along the diameter and any islands or separatrices are much larger than the space between individual points [18]. As an example, Figure 13 shows an orbit from the KAM book that typifies the assumptions it makes. The figure shows a typical separatrix orbit from the Henon map. The orbit contains dense regions near the cross points compared to points further along the diameter. The lobes are significantly wider than the inter-point distance along the diameter. This large relative distance explains why the MST can effectively capture the structure of the orbit with as few as 150 points. Figure 2, however, shows that the separatrix from our dataset has branches that cross the lobes and indicates that lobes approach the size of the inter-points distances. With few points, the MST incorrectly groups these points together rather than apart. Since the orbits in our data appear to violate the assumptions of the original algorithm, we must manipulate thresholds such as the percent primary nodes, hyperbolic line length, and branch lengths to try and alleviate these problems.

In the Henon map orbits of Figure 13, one is able to isolate a single vertex that joins the two lobes of the separatrix; thus, the orbit has a single identifiable cross point. With few points, slight perturbations in the positions of the points would not eliminate this cross point. With many points, the cross point is still obvious. In the separatrix from our dataset, however, one identifies cross points only as dense regions rather than individual points. From the perspective of KAM, the cross points are very noisy and contain many very short branches. Also, KAM identifies the upper segment of the orbit in the figure as the cross point because this fits the assumptions. From these figures, it would appear that for our dataset, KAM only identifies cross points nearby their true location.

Table 7 describes some early results of the evaluation of the classifiers on previously unseen data. Our classifiers are currently trained on a small set of orbits. However, we expect that in practice, we will need to classify orbits where there is a lot of variation in each class. This validation test set is an attempt to determine how robust our classifiers would be to such variation. As mentioned earlier, this validation set

consists of 100 orbits and is mainly composed of islands and separatrices. As in previous experiments, we observe that the default KAM classifier performs quite badly and though the custom KAM classifier improves the accuracy, it is still quite poor compared with the standard classifiers. Moreover, using a large number of points for the orbits in the training set helps the accuracy of all the classifiers except the default and custom KAM classifiers. These validation experiments are in their early stages and additional experiments with a larger validation set will need to be performed before we can make any conclusive statements about the performance of different classifiers.

3.7 Summary of Single-Orbit Classification

In single-orbit classification, the objective is to extract features from individual orbits to correctly identify their class. We implemented and extended an existing orbit-classification algorithm called KAM (also referred to as the default KAM classifier in this report). We used it to extract features from the orbits and compared the default and our customized version of KAM to several feature-based classification algorithms. The results show that the accuracy is relatively stable as the number of points in an orbit decreases. Accuracy improves when the algorithm can train and test on orbits with the same number of points. The results also show that the default KAM algorithm performs very badly on our dataset but that our customized version is comparable to the other algorithms. The poor performance of the KAM algorithm is best explained as the result of applying an algorithm to a dataset for which it was not designed.

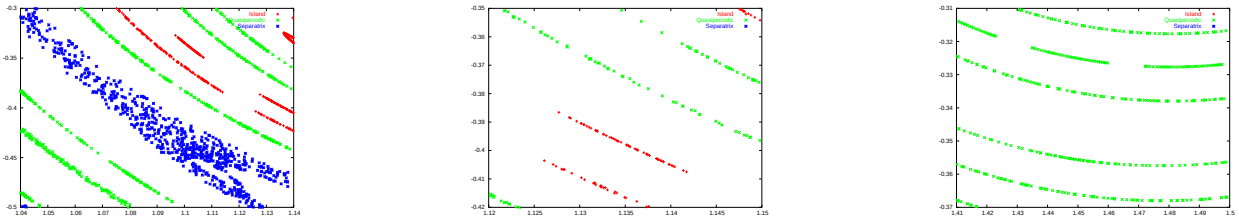
As part of the larger analysis problem, our conclusion for single-orbit classification is that one should first use KAM to generate features of the orbit and then apply a classification algorithm such as the nearest-neighbor or the decision tree for ultimate classification.

4 Multi-Orbit Classification

As described in Section 2.3, a solution to the multi-orbit classification problem must perform two tasks. First, it must identify the components of the plot that correspond to regions of interest to physicists such as islands and separatrix cross points. Second, it must use the component-level information to form a description of the entire multi-orbit plot that one can later use to compare with other plots from both simulations and experiments. We have made some initial progress toward these goal by classifying and clustering points rather than orbits.

At first glance, one can view multi-orbit classification as a repeated single-orbit classification problem. When orbits are available separately, we can classify each orbit, extract its components, and combine them to describe the entire multi-orbit plot. Indeed, this is the approach of the KAM expert system [18]. Unfortunately, in our case, the problem is further complicated by the fact that orbits are not available separately for experimental data as they are for the simulations. This means that any single-orbit classification algorithms that we develop for simulation data cannot be readily applied to the multi-orbit data from the experiments. Algorithms specifically designed for multi-orbit data can, however, be readily applied to multi-orbit plots generated from single-orbit data. We can therefore solve the more complex multi-orbit classification problem for both experimental and simulation data by concentrating on algorithms that do not require individual orbits.

Abandoning the reliance on individual orbits requires relaxing the analysis requirements. Our initial approach does not try to find individual orbits. Instead, we try to identify as many interesting components as possible in the multi-orbit plot and decide later what to do with them. We propose, therefore, to identify the components as a preprocessing step to the further characterization of multi-orbit plots. This section highlights some of the ideas we have tried in this first identification step.



(a) Separatrix (b) Island (c) Quasi-periodic

Figure 14: Equally sized neighborhoods of a point from each class in the orbits from Figure 3.

4.1 An Initial Approach

Although multi-orbit data does not include information identifying the individual orbits, it does include a large number of points, some of which constitute the interesting components. Instead of orbit classification and graph processing, we consider the multi-orbit problem is one of point classification and point processing; that is, we process the points and assign class labels to them.

To identify the interesting components of multi-orbit plots, we apply the following process:

1. Using the individually available orbits from simulations, construct multi-orbit plots by super-imposing the individual orbits onto a single plot. Using our 600 orbits, we have 6 multi-orbit plots that resemble Figure 3.
2. Extract point-level features (Section 4.1.1) for all points in all plots. The class label for a point is the class of the orbit that generates it.
3. Split the 6 plots into a training and testing set of 5 and 1 plots, respectively. For example, we use Figure 3 as the testing plot in this section.
4. Train a classifier on the training plots.
5. Test the classifier on the testing plot recording the label it assigns to each point.
6. Perform density-based clustering separately on points labeled as separatrix and island.
7. The components are the clusters identified by the clustering algorithm.
8. The convex hull of the clusters can be used to compute a first approximation to the geometric description of the components.

4.1.1 Point-Level Features

The point-level features attempt to convey the relationship between a point and the points near it in the plot. The intuitive justification for this choice of features is that both islands and separatrix components are visually very different from the points along a quasi-periodic orbit. As illustrated in Figure 14, separatrix cross points are dense but occupy a region which is wider than quasi-periodic orbits. Islands appear to have discontinuities; thus, the empty regions around them and their tips are very distinct from the other two classes. Also, the middle sections of islands and points outside cross points on separatrices are generally indistinguishable from quasi-periodic orbits. Quasi-periodic orbits appear similar at all orientations.

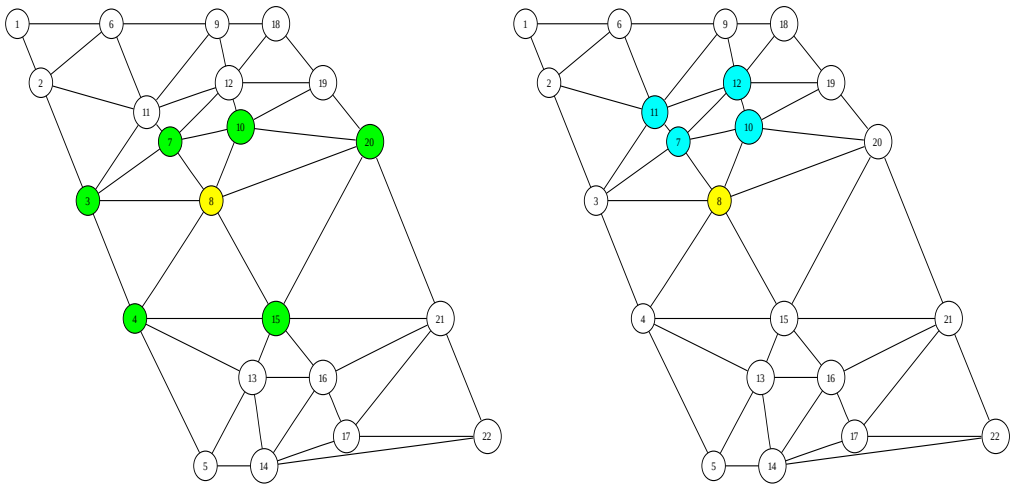


Figure 15: Delaunay and local neighborhoods (shaded) with respect to vertex 8.

To capture this relational information, the feature extraction algorithm obtains statistical information about the points in the neighborhood of a point. It first computes the Delaunay triangulation of the points in the plot using the previously defined Euclidean distance between points. The result of the triangulation is a graph that has edges between points and their neighbors. Specifically, the nearest neighbor of a point is the closest adjacent vertex to the point. Although the triangulation guarantees finding the nearest neighbor, it does not guarantee that the kth nearest neighbor is the kth closest adjacent vertex in the Delaunay graph. We can, however, use the Delaunay graph to define neighborhoods given the previously defined vertex-level distance function and the fact that no vertex has an edge to itself in the Delaunay triangulation.

Delaunay Neighborhood Given the Delaunay triangulation graph $D = (V, E)$, the set of vertices $N_D(p)$ is the Delaunay neighborhood of a point p with radius $r_D(p)$ if and only if:

$$\begin{aligned} N_D(p) &= \{q \mid adj(p, q)\} \\ r_D(p) &= \frac{1}{|N_D(p)|} \sum_{q \in N_D(p)} length(\langle p, q \rangle) \end{aligned}$$

The neighborhood $N_D(p)$ is the set of vertices in the graph adjacent to p. The radius $r_D(p)$ is the average length of the edges.

Local Neighborhood The local neighborhood attempts to capture a more symmetric locality. Given the Delaunay neighborhood $N_D(p)$ of a point p, the local neighborhood of a point p is the set of vertices $N_L(p)$ if and only if

$$N_L(p) = \{q \mid q \in V \wedge d(p, q) \leq r_D(p)\}$$

Figure 15 shows examples of the Delaunay and local neighborhoods for a small section of an orbit graph. The graph shows a portion of two islands in an island chain. Vertices 8 and 15 are the respective bottom and top of two islands. The island chain is between an orbit on the left containing vertices 1–5 and an orbit on the right containing vertices 18–22. The Delaunay neighborhood includes vertices from the left and right orbits and the other island. Intuitively, we would want the neighborhood of a vertex v to contain vertices

in the same orbit as v. The local neighborhood better captures this idea of neighboring points in that it requires points to be close in distance rather than adjacent in the Delaunay graph. In the figures, vertices 11 and 12 have been added to the local neighborhood because they are within the radius. Vertices 3, 4, 15, and 20 have been removed from the local neighborhood because they are much further than the radius.

We next extract the following statistical and shape-related features using the Delaunay and local neighborhoods.

Statistical Features The following features depend on the local neighborhood of a point p:

$|N_L(p)|$ The number of points in the local neighborhood.

$r_D(p)$ The radius of the Delaunay neighborhood.

σ_{weight} Standard deviation of the edges in the Delaunay neighborhood.

$|N_D(p)|$ Number of edges in the Delaunay neighborhood.

$\rho(N_L(p))$ Density of the local neighborhood: $\frac{|N_L(p)|}{r_D(p)}$.

Spatial

$d(p, nn(p))$ Distance between p and its nearest neighbor where the nearest neighbor is $nn(p)$ such that:

$$\langle p, nn(p)\rangle \quad = \quad \arg \min_{\langle p,q\rangle \in N_D(p)} d(p, q)$$

This is equivalent to the closest vertex in the Delaunay neighborhood.

$\theta(p, nn(p))$ Angle between p and its nearest neighbor:

$$\theta(p, nn(p)) \quad = \quad \arctan \frac{d_2(p, nn(p))}{d_1(p, nn(p)}$$

where d_i is the absolute value of the difference between p and q with respect to the ith coordinate.

Linearity Number of points within $\epsilon = 0.005$ of the line between p and its nearest neighbor within the local neighborhood.

The spatial features are designed to characterize the spatial relationship among the points in the neighborhood. The nearest neighbor features are based on the observation that points in an orbit usually appear near each other; thus, we expect that the nearest neighbor to a point is on the same orbit. The linearity feature is based on observations on neighborhoods such as those shown in Figure 14. From these neighborhoods, middle regions of islands and quasi-periodic orbits resemble line segments. Separatrix cross points are difficult to characterize by a line; they more closely resemble regions or points.

Shape Context

Histogram The histogram of the log-distance and angle between the point and every point in its local neighborhood. We use 4 bins each for the distance and angle values so that the histogram constitutes 16 features in the vector.

$\min_\theta$ The minimum angle between the point and any other point in its local neighborhood.

$\max_\theta$ The maximum angle between the point and any other point in its local neighborhood.

$med_{r,\theta}$ Median values for the log-radius and angle for the most likely distance and angle.

$\max_p$ Probability of the most likely radius-angle pair.

χ^2_R The result of the χ^2 test between the distribution of the histogram and a uniform distribution over the values in the histogram. The test is defined as:

$$\chi^2_R = \sum_{i=1}^{4}\sum_{j=1}^{4} \frac{\left(p_{i,j} - \frac{1}{16}\right)^2}{p_{i,j} + \frac{1}{16}}$$

where the histogram contains 16 bins in total. The χ^2 test has been used to compare shape contexts for shape matching [4]. The test returns a measure of how different the distribution of radius and angle values is from a uniform distribution wherein all values in the histogram would be the same.

The shape context features are based on those used for matching shapes in the field of computer vision [4]. In the original paper on shape matching, the shape context uses the distribution of the distances between one point and all other points in the shape. This would not be useful for multi-orbit classification because the relative positions of the orbits do not hold for different orbits. For example, an island on the left of the point in one plot may be on the right in another plot. At the local neighborhood level, however, the ends and sides of islands do have distinct shapes that remain distinct for different plots. Hence, we extract the shape context features in a local neighborhood of a point.

4.1.2 Classifying Points

After constructing these 31 features for each point in the plot, the algorithm first trains and then tests a classifier on subsets of the multi-orbit plots. We use the decision tree classifier for the experiments. Although we rarely have a testing accuracy greater than 50%, we are not really interested in correctly classifying all the points–just the important ones.

Upon examination of the testing results, we find that the classifier can correctly label individual separatrix cross points and islands. It cannot effectively distinguish between quasi-periodic orbits and those that are locally similar but belong to a different class.

Since we are only interested in the orbit components, the next step is to ignore the points labeled as quasi-periodic and separate the multi-orbit plot into one plot each for the separatrix and island classes. On these separated plots, we perform the component identification processes. Figure 16 shows the combined separatrix and island chain classes after testing the classifier. Some of the straight parts of the orbits are missing in these plots because they were labeled as quasi-periodic. The individual islands and separatrix cross points are still visible in the figure.

4.1.3 Identifying Components

To identify components from class-separated multi-orbit plots, we apply a density-based clustering algorithm. Unlike partitional clustering algorithms, these algorithms requires a density parameter rather than the number of clusters. Given the distance-based density parameter, ϵ, the density-based clustering algorithm DBScan finds clusters such that the points in the clusters are either within ϵ of each other or near points that are within ϵ of points [9].

The result of the clustering algorithm is a set of clusters that is mutually exclusive but not necessarily exhaustive. Some points may be unclustered because they are too far from other points with respect to the density parameter. Since cross points are much more dense than the points that border the lobes, cross points should correspond to dense clusters. Similarly, islands are dense regions around the orbit and should also be in dense clusters. Points not in the components should appear to be outliers; thus, the clustering appears to satisfy our requirements.

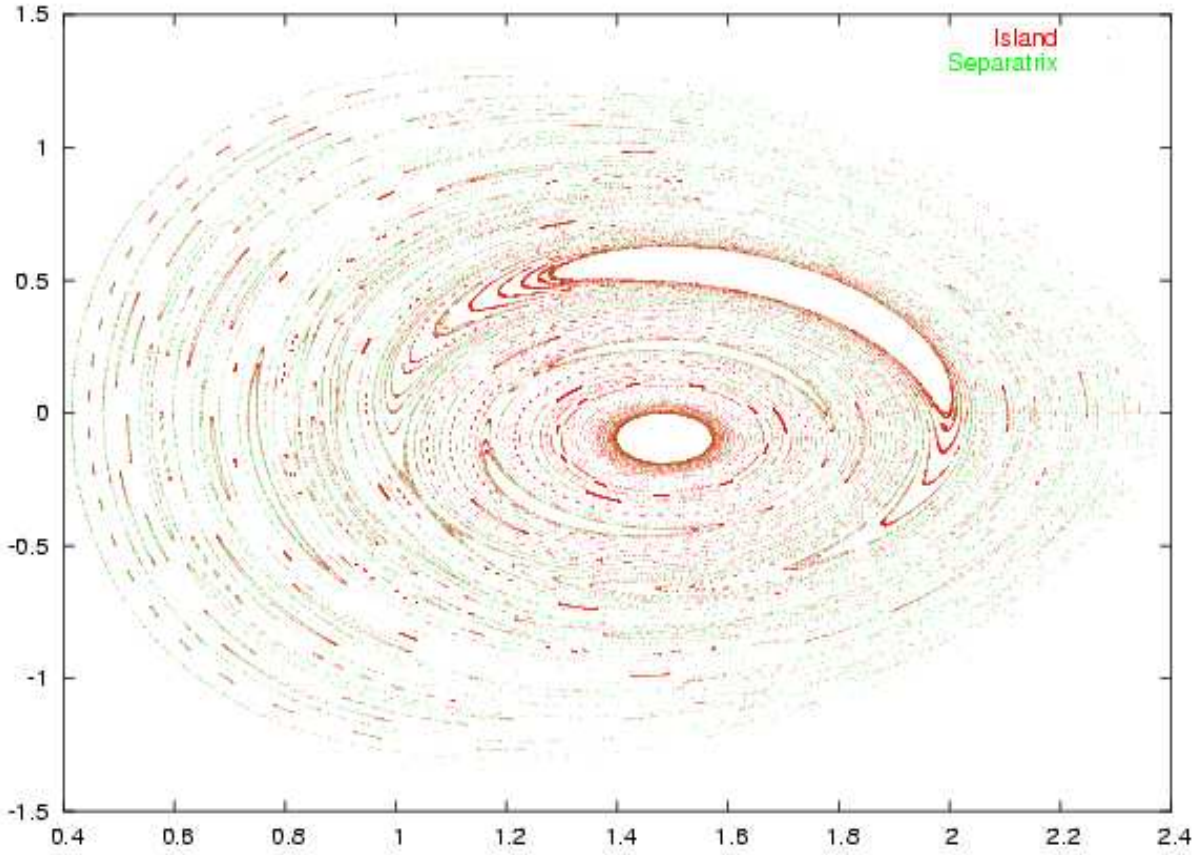


Figure 16: Separate classes from the multi-orbit plot in Figure 3.

After clustering, we identify the shape of the component as the convex hull of the points in the cluster. The convex hull provides an outline of the shape of the component. We can use the hull to calculate a first approximation to the width, height, and centers of the components to support our analysis objectives.

4.2 Early Results

Figure 17 shows the result of applying the component identification process to the plot from Figure 3. The classifier was trained on at most 1000 points from each of the 100 orbits from five of the size multi-orbit plots and tested on the remaining plot from Figure 3.

The clustering algorithm correctly identifies most of the islands and a few of the separatrix cross points. It finds most of the islands but merges some of them. It does correctly identify dense separatrix regions, but Figure 17 shows that the algorithm finds many small clusters near the cross point instead of one large cluster. This is likely the result of a too-restrictive density parameter. Also, it incorrectly finds cross points on islands, but this is probably due to the fact that the large islands in the center of the plot resemble separatrix lobes at the neighborhood level.

The most difficult aspect of the process is selecting the right parameters for the clustering algorithm. The figures show the clustering results with $\epsilon_I = 0.003$ and $\epsilon_S = 0.008$. The parameters are the result of trial and error, but the original paper offers some help for parameter selection [9]. The suggested method is to compute the 4-nearest-neighbor distance and use the "first value of the first valley" [9] as the value of the parameter. Following this advice, we tried several values and found that the clustering oscillated from all points belonging to one cluster and no points belonging to any cluster. Although we do have some success with the values, we have as yet found no automated selection process. In a later paper, however, the authors introduce the OPTICS algorithm that is specifically designed to assist with the parameter selection problem, but we did not have access to a full implementation [1, 16].

5 Problem Revisited

The goal of automated analysis is still somewhat distant, but the techniques presented in this report provide the basis for a preliminary outline of a possible approach to the analysis. For simulation data, where the points belonging to each orbit are explicitly available, we can first classify each orbit, assigning a class label of quasi-periodic, separatrix, islands, or unknown. This is done by extracting features from the orbit and using either a rule-based or a feature-based classifier. In our work, we extracted features based on the ones described in the KAM book [18], though other options are certainly possible. We used the default KAM and the custom KAM as the rule-based classifiers, as well as several feature-based classifiers including decision trees and support vector machines. Once each orbit is assigned a class label, we can do further processing to extract the components of interest including the width of the separatrix lobes, the locations and widths of the islands, etc. These components extracted for each orbit in the plot will collectively describe the plot.

In experimental data, we are provided all the orbits in one plot, without an explicit assignment of points to orbits. In this case, which we refer to as the multi-orbit case, we first considered a solution approach based on the classification of the points and then attempted to group the points belonging to a single class to see if we could find approximations to the components of interest. The end goal was to directly find these components without first finding the orbits or assigning class labels to them. We showed some early progress in addressing this rather difficult problem, one which is made more challenging as the number of points per orbit in experimental data will be quite small, of the order of 100.

Our ultimate goal is to create a description for each plot generated by either simulations or experiments. In addition to the early work described in this report, we will also consider other possibilities including an inter-orbit reasoning system such as the one described in the KAM book [Yip91KAM] and the ideas described in the related work of Section 6. These descriptions can then be used by physicists in their analysis of orbit data.

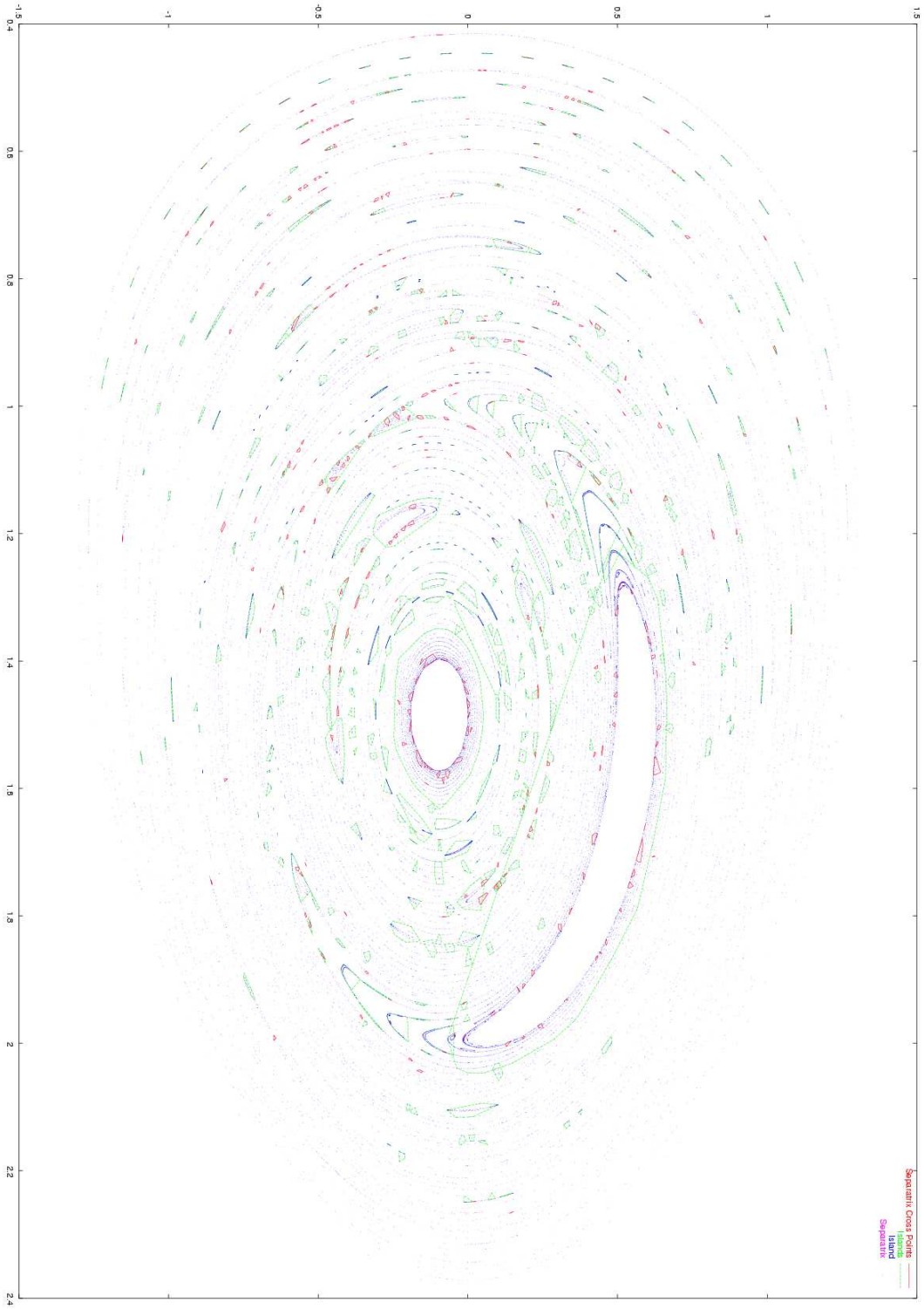


Figure 17: Identification of components in multi-orbit plot.

6 Related Work

Based on the literature from the physics and machine learning fields, orbit classification has two very different meanings. In physics, the objective is to move from knowledge of the physics of particles to classes of orbits. In machine learning, the objective is to infer the class information from the data typically in absence of any domain knowledge. Unfortunately, the results from one field are not particularly helpful for the other.

6.1 Orbit Classification in Physics Literature

In the area of orbit classification from the physics literature, researchers analyze the physical equations that describe particle movements and use this knowledge to explain and identify different classes that can arise in the experiments. The equations that govern particle orbits depend on a set of parameters. Given the parameters and equations, one can simulate the particle orbits, and observe the different classes. One method of orbit classification is to classify the orbits in the parameter space of their equations [5, 10, 19]. A related method is to extract features on the parameter space and the equations such as a popular method called Constants-of-Motion (COM) that uses invariants present in the equations of the particle orbits [13]. Orbits of the same class trace out continuous trajectories in the COM parameter space, and researchers have applied analytical and geometric approaches to model the orbit likely to result given the parameter values [7, 3, 14].

Although the classification of orbits given their equations and parameters is one of the fundamental objectives of the physics research, these results do not aid the problem that is the subject of this study. This is because the equations and the parameters are either unknown or unavailable to the classification algorithm. Work in the physics literature that can aid our problem is research that fits the parameters to the data. Work in this area uses learning algorithms to identify the parameters that are likely to explain the resulting orbit data. Researchers have tried classification algorithms to learn to discriminate between high- and low-mode plasma states [6]. Although the algorithm operates in the space of parameters, the true state of all examples is not known, *a priori*. A more related work applies multilayer perceptrons and function parameterization algorithms to directly learn the parameters given a database of orbits [15]. Finding the parameters from the data allows the physicists to use the existing classification methods that require known parameters.

6.2 Orbit Classification in the Machine Learning Literature

In contrast to work in orbit classification from the physics literature, the subject of this study requires classification of the data without knowledge of the equations or their parameters. In machine learning literature, pattern classification researchers look for algorithms that can learn to distinguish between examples that arise from different classes or "states of nature" [8]. Here, the objective is to discover novel and useful classification algorithms rather than the states of nature themselves.

Despite the difficulty of the problem and progress that this paper shows, orbit classification has received relatively little attention in the machine learning literature. Much of the work that does exist uses parameter information from nonlinear differential equations to map out the phase space of a system.

Many of these early approaches are expert systems in the domain of numerical experimentation in nonlinear systems. One of the earliest approaches is KAM whose primary objective is to discover the complete phase space of nonlinear systems [18]. To accomplish this goal, it generates new orbits by manipulating the parameters to the equations. A key component to KAM and the basis for our experimentation is the KAM classifier in which the author applied domain knowledge to derive a set of static rules. Although KAM is not technically a learning algorithm, it is one of the earliest algorithmic approaches to classifying orbits without direct knowledge of the parameters.

The KAM method belongs to the paradigm known as spatial aggregation [20, 17], which is based on visual reasoning. Starting with an image-like input (e.g. the points of an orbit), the idea is to transform

the point representation into more economical symbolic representations using techniques similar to those in computer vision. Such techniques are also used to manipulate topological structures in spatial data. The early work in the use of geometric and spatial reasoning includes not only the KAM approach [18] used in this paper, but also the MAPS system, which designs control laws based on a geometric analysis of the state equations of a dynamical system [20].

More recently, the area of Qualitative Spatial Reasoning [2] has evolved to provide a solution to problems where numerical methods cannot describe the geometric and topological structures in data sets required to answer high-level spatial queries. This idea of spatial reasoning is broadly applicable to problems in scientific domains such as geographic information systems, metereological and fluid flow analysis, and CAD systems. Other related work includes the application of classification as part of a nonlinear orbit expert system [11] and the use of the density of connected components to identify connected regions in fractal maps for similarity search [12].

Little research has been done in the area of multi-orbit classification. Returning again to the expert systems literature, KAM and others use parameter space information to characterize important components in multiple graphs [18, 11]. Their approach is to successively generate orbits with different parameters and use the relation between the classes of orbits and the parameters to capture a complete phase portrait of the system. This is not immediately useful in our work because we have neither individual orbits nor a parameter space. Some of the the inter-orbit rules may, however, prove to be useful.

7 Conclusion

This report presents preliminary work toward an automated data-analysis toolkit supporting experiments and simulations from prototype fusion devices. We focus on the analysis of Poincaré plots. A plot consists of several orbits each corresponding to a particle moving in the device. The positions of the particle trace out a shape that belongs to one of three known classes: separatrix, island chain, or quasi-periodic. The focus of our work in this paper is on classification techniques that can infer the class label and identify interesting components given individual or multiple orbits in a plot.

In this preliminary work, we have made some progress toward the eventual goal of an automated analysis and interpretation tool. The goals of the study were to study single-orbit classification, investigate their accuracy and tolerance for orbits with few points, and then to investigate the identification and extraction of components for multi-orbit data.

For single-orbit classification, we implemented and extended the orbit-classification algorithm KAM. The default KAM classifier applies a set of static rules on features extracted from the Euclidean minimal spanning tree of the points in a single orbit. We extended both the rules and features to create a customized KAM classifier and features for standard pattern classification algorithms. The results show that the existing classification algorithms like the k nearest neighbor and the decision tree outperform both the default and our customized KAM classifier. They have high accuracy and good tolerance for orbits with as few as 100 points. Additionally, they are sufficiently straightforward to implement in the analysis pipeline.

Compared to single-orbit classification, multi-orbit data appears to be more difficult. Despite this difficulty, we have some progress with a combination of classification and clustering. First, we train a classification algorithm on known training multi-orbit plots and test it on an untrained plot to label the individual points based on the orbit that generated them. Next, we apply density based clustering separately to the separatrix and island points. Components are identified as the dense clusters and we use the convex hull to mark their extent. Applying this extraction, classification, and clustering process shows visually valid results.

Future work in the analysis area will concentrate on evaluation of more sophisticated single-orbit features, multi-orbit results and better component extraction methods. More objective evaluation metrics are necessary to compare this technique to other classification techniques and validation from the training data. Supervised clustering that performs clustering using the class information may be useful in correctly identifying components. We expect that reasoning systems for orbit components will be necessary for successful

automated interpretation of orbit data.

Acknowledgments UCRL-TR-215690: This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

We gratefully acknowledge the domain expertise provided by Neil Pomphrey and Don Monticello at the Princeton Plasma Physics Laboratory. We would also like to acknowledge Scott Klasky for introducing us to this problem and for his support of this work, including the access to the datasets.

Abraham Bagherjeiran is a graduate student at the University of Houston. This work was done when he was a student intern at LLNL during summer 2005.

References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, 1999.

[2] C. Bailey-Kellogg and F. Zhao. Qualitative spatial reasoning: extracting and reasoning with spatial aggregates. *AI Magazine*, 24(4):47–60, 2004.

[3] V. S. Belikov and Yu V. Yakovenko. Classification of particle orbits in high-β spherical tokamaks. *Physics of Plasmas*, 8(10):4501–4508, 2001.

[4] Serge Belongie, Jitendra Malik, and Jan Puzicha. Matching shapes. In *8th IEEE International Conference on Computer Vision*, pages 454–463, 2001.

[5] W. G. F. Core. Non-standard particle orbits in a thermonuclear tokamak plasma. *Nuclear Fusion*, 40(10):1715–1719, 2000.

[6] R. D. Deranian, R. J. Groebner, and D. T. Pham. Use of a pattern recognition algorithm to obtain a parameterization of low-mode and high-mode plasma states. *Physics of Plasmas*, 9(6):2667–2674, 2002.

[7] C. M. Doloc and G. Martin. A topological approach to the problem of charged particle trajectories in a toroidal axisymmetric configuration. *Physics of Plasmas*, 2(10):3655–3666, 1995.

[8] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience, 2001.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

[10] C. T. Hsu and D. J. Sigmar. Alpha-particle losses from toroidicity-induced alfvén eigenmodes. Part I: Phase-space topology of energetic particle orbits in tokamak plasma. *Physics of Fluids B*, 4(5):1492–1505, 1992.

[11] Wood W. Lee and Benjamin J. Kuipers. A qualitative method to construct phase portraits. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 814–819. MIT Press, 1993.

[12] V. Robins, J. D. Meiss, and E. Bradley. Computing connectedness: Disconnectedness and discreteness. *Physica D*, 139:276–300, 2000.

[13] J. A. Rome and Y. K. M. Peng. The topology of tokamak orbits. *Nuclear Fusion*, 19(9):1193–1205, 1979.

[14] S. Satake, H. Sugama, M. Okamoto, and M. Waktatani. Classification of paritcle orbits near the magnetic axis in a tokamak by using constants of motion. Technical Report NIFS-677, National Institute for Fusion Science, 2001.

[15] A. Sengupta, P. J. McCarthy, J. Geiger, and A. Werner. Fast recovery of vacuum magnetic configuration of the W7-X stellarator using function parameterization and artificial neural networks. *Nuclear Fusion*, 44:1176–1188, 2004.

[16] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[17] K. Yip and F. Zhao. Spatial aggregation: Theory and applications. *Journal of Artificial Intelligence Research*, 5:1–26, 1996.

[18] Kenneth Man-Kam Yip. *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press, 1991.

[19] Hayakawa Yumi, Takahashi Toshiki, and Kondoh Yoshiomi. Classification of particle orbits and related stochasticity of plasma ion motion in a field-reversed configuration with D $-^3$He advanced fuel. *Nuclear Fusion*, 42:1075–1084, 2002.

[20] F. Zhao. Extracting and representing qualitative behaviors of complex systems in phase space. *Artificial Intelligence*, 69:51–92, 1994.