UCRL-CONF-215802

LAWRENCE LIVERMORE NATIONAL LABORATORY

Graph-based Methods for Orbit Classification

A. Bagherjeiran, C. Kamath

October 3, 2005

SIAM International Conference on Data Mining
Bethesda, MD, United States
April 20, 2006 through April 22, 2006

Graph-based Methods for Orbit Classification

Abraham Bagherjeiran, Chandrika Kamath [*]

Abstract

An important step in the quest for low-cost fusion power is the ability to perform and analyze experiments in prototype fusion reactors. One of the tasks in the analysis of experimental data is the classification of orbits in Poincaré plots. These plots are generated by the particles in a fusion reactor as they move within the toroidal device. In this paper, we describe the use of graph-based methods to extract features from orbits. These features are then used to classify the orbits into several categories. Our results show that existing machine learning algorithms are successful in classifying orbits with few points, a situation which can arise in data from experiments.

Keywords: orbit, Poincaré plot, classification.

1 Introduction

The quest for low-cost fusion power has led to the construction of devices such as the National Compact Stellarator Experiment (NCSX) at the Princeton Plasma Physics Laboratory (PPPL). These devices allow physicists to perform magnetic confinement experiments which 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 possible the simulation of these experiments computationally in three dimensions over time. These simulations allow the physicists to design new reactors and select the parameters to be used in experiments. The results from the experiments are, in turn, used to validate the simulations. Thus, the analysis of the data from both simulations and experiments is a key step in the understanding and development of fusion reactors.

In this paper, 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 where each point represents the intersection of the particle with the plane as it goes around the device. The positions of the points in an orbit describe a particular shape, based on which we can assign a class label to an orbit. Once a class label has been assigned to an orbit, it can be analyzed further to extract additional characteristics of interest.

In this paper, we describe how we can use graph-based techniques to extract features describing an orbit. These features are then used in both rule-based and feature-based learning algorithms to assign class labels to orbits. This paper is organized as follows. Section 2 provides an overview of the problem of orbit classification. Next, in Section 3, we discuss the properties extracted from each orbit by considering it as a graph. In Section 4, we describe how we can modify the features obtained in an existing rule-based approach, called KAM [7], so that it can be applied to our datasets. We also describe the extraction of additional features from the data to improve the representation of an orbit. Section 5 illustrates the discriminative ability of the features. In Section 6, we show that our additional features result in higher classification accuracy. We discuss related work in Section 7 and conclude the paper in Section 8 with a summary and discussion of ideas for future work.

2 Problem Definition

Figure 1 shows the schematic of the NCSX. 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.

Depending on the shape of the orbit, it can be assigned a class label. Figure 2 depicts three different orbits–a separatrix, an island chain, and a quasi-periodic orbit. There is also an additional class of stochastic orbits, which we will not consider in the present analysis. From these examples, we see that the orbits are visually very different. Notably, the quasi-

[*]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA

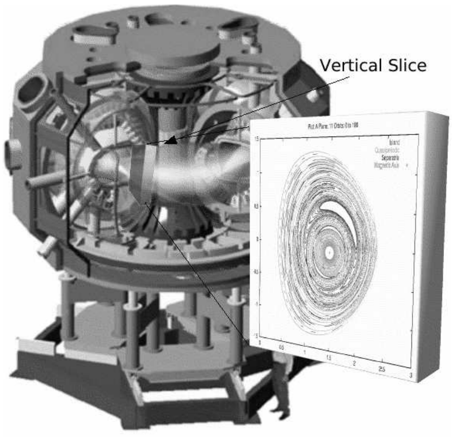

Figure 1: A schematic of the NCSX reactor currently under construction at the Princeton Plasma Physics Laboratory. Inset shows a plane perpendicular to the magnetic axis illustrating the intersections of the particles.

periodic orbit appears to be a closed curve, with no width to the curve. The island chain orbit has two distinct islands in this example. The separatrix orbit, like the quasi-periodic orbit, appears closed but has radial gaps called lobes; there are two such lobes, one on the top and the other on the bottom, in this orbit. The separatrix orbit can be considered to be one where there are several islands, whose tips merge, end-to-end. The region of dense points where the tips merge (in our example, there are two such points, one on the left and the other on the right) are called cross points.

Typically, all the orbits on a plane are provided together in what is referred to as a "puncture plot" or a "Poincaré plot", as in Figure 3. When the orbits are considered together in a plot, there is additional analysis which can be performed by considering the relationships between orbits. However, in this paper, we will focus on the classification of a single orbit, considered by itself.

Orbits resulting from computer simulations usually consist of a thousand or more points, while those from experiments consist of 50 to 100 points. When the number of points is small, the correct classification of an orbit can be a challenge as there may not be enough information in the data to identify the shape. In our work, we will consider how the accuracy of a classifier changes as the number of points in an orbit is reduced. If we can correctly classify orbits using a few points, it reduces the time for the extraction of features, leading to a faster turnaround in the analysis. Further, data from simulations contain floating point errors which build up over time, so that it is more accurate to work

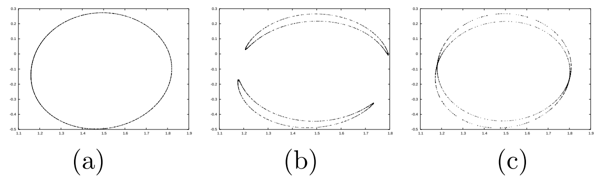

Figure 2: An example of an orbit from each class. (a) Quasi-periodic. (b) Island chain with two islands. (c) Separatrix with two cross points.

with the first few points.

3 Orbit Preprocessing

In this section, we discuss how we can extract features from an orbit by considering its graph representation, as described in [7]. We first 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 Definitions from Graph Theory

We 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.

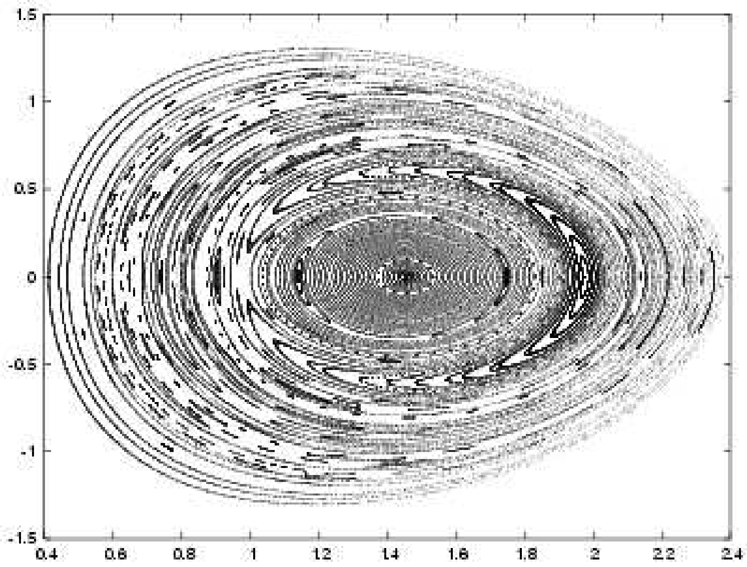

Figure 3: An example of a puncture plot. This plot contains 100 orbits, with the magnetic axis located at approximately x = 1.0, y= 0.

Edge Properties An edge, e, has a real-valued property $length(e) \in \mathbb{R}$ equal to the Euclidean distance between its vertices. We sometimes refer to this as the *weight* of an edge.

3.2 Generating the Orbit Graph The first preprocessing step is to compute a graph representation of an orbit in the form of its Euclidean Minimal Spanning Tree (MST). 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 that it characterizes the shape of the orbit and is similar to the way humans see objects [7]. For the remainder of this paper, we will refer to the MST of the orbit as the orbit itself or the orbit graph.

3.3 Partitioning the Graph The next step is to partition the graph into independent clusters by removing edges. The algorithm used in our implementation removes all edges whose weight exceeds a threshold $t_{partition}$. In other words, edges connecting vertices which are more than $t_{partition}$ apart in Euclidean distance are removed, yielding k 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.

3.4 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. The properties described next 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. To compute the diameter of the MST, 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 $false$. The algorithm also identifies the beginning and end points of the diameter for later use in feature extraction.

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 4, 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. Given thresholds $0 < t_{shallow} < t_{deep}$, $shallowBranch(v)$ is defined as:

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

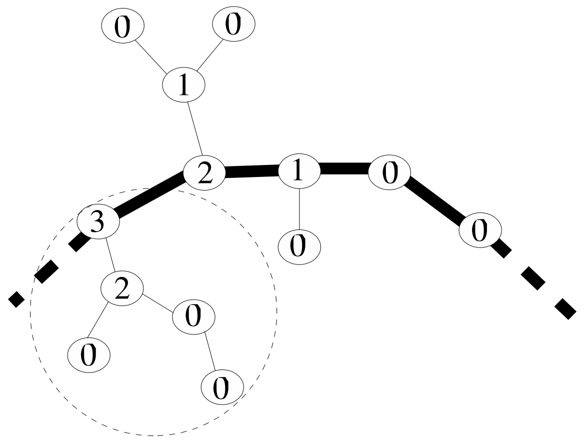

Figure 4: Branch calculation example on a small portion of a larger graph. The thick edges indicate the diameter of the graph. Vertex labels show the value for the *branchLength* property.

and $deepBranch(v)$ is defined as:

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

The $primaryBranch(v)$ property

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

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 4, 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 $primary(v)$ property defined as:

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

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 and is specifically designed to describe separatrix orbits. Figure 5 shows an example of the ideal cross point of a separatrix orbit, with two deep branch vertices. 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 5, the line segment sequence is

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

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*, *shortLine*, *deepBranch* at least twice. 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.

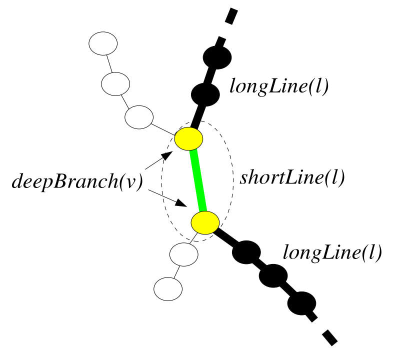

Figure 5: Example of hyperbolic structure for a portion of a separatrix orbit. The thick lines and filled circles indicate the diameter of the graph. The circled region is the cross point.

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

Table 1: Properties extracted from the orbit graph and their corresponding thresholds.

3.5 Summary of Orbit Pre-processing As shown in Table 1, we extract several vertex-, edge-, and graph-level properties from the minimal spanning tree of the points in the orbit. These properties are then used to derive several features for use in various classifiers.

4 Features and Classification Algorithms

We next use the properties described in the previous section to create the features for classification algorithms. We consider the rule-based approach of [7], which we call the default KAM, a customized version of KAM which has been modified for our data, 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.

- f_X: A feature X 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.

4.1 The Default KAM classifier For the default KAM classifier, we first extract the features described in the book [7] and then apply a set of static rules on 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.
- $n_{clusters}$: 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 2.
- *Hyperbolic*: Indicates if the graph has a hyperbolic structure. Obviously, separatrices are the only class of orbits where this feature should be true.
- *Origin*: Indicates if 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 2 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 as follows:

- We use variable parameters instead of hard-coding the threshold values as done in KAM.

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

- 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 [7] 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 were interested in seeing if a decision tree would learn similar rules. So, using a training set with 600 orbits, where the class label had been assigned using KAM, we created the tree shown in Figure 6. This decision tree learns the KAM predications to over 99% accuracy on the training set. Interestingly, the decision-tree rules have some similarity to the original KAM rules.

- 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.

4.1.1 Selection of Thresholds

Although the default KAM classifier discussed here uses the parameters from the KAM book as shown in Table 2, it does not

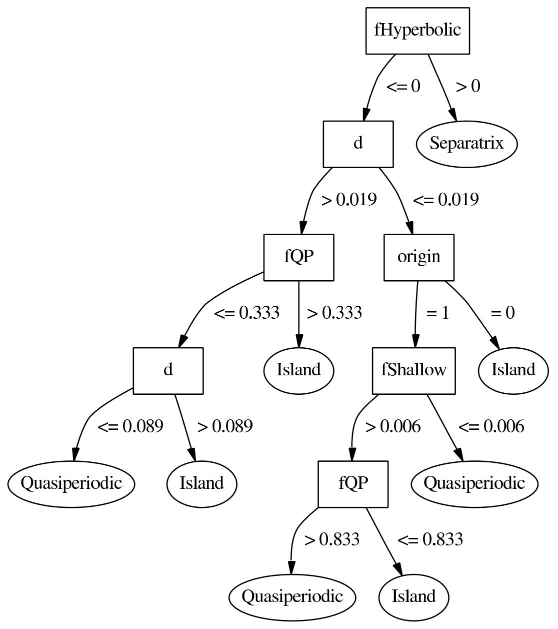

Figure 6: Decision tree trained to predict the KAM evaluation.

recommend values for the thresholds shown in Table 1. We use a class-conditional probability distribution approach to find these thresholds. 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.

4.2 The Custom KAM classifier An analysis of the performance of default KAM on our dataset indicated that we could improve the results by some simple additions. Our extension to KAM, called custom KAM, also uses static rules, and in addition to all the features of the default KAM classifier, includes one additional feature with a 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. We also allow the user to change both the parameters and the thresholds. 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.

4.2.1 Selection of Thresholds
In the custom KAM classifier, where both the parameters and the thresholds can be changed, 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 use a genetic algorithm to search this multi-modal threshold parameter space [1].

The genetic algorithm manipulates the thresholds from Table 1, the parameters for the default KAM classifier in Table 2, and the t_{LS} threshold for the custom KAM classifier. The fitness of an individual is the average similarity between idealized feature vectors for each orbit class and the extracted feature vectors of a few example orbits.

4.3 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 as well as features that convey the underlying statistics of the edges and clusters of the orbits. As we will demonstrate in Section 5, 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.
- K_I: Island predicate evaluation.
- K_S: Separatrix predicate evaluation.
- K_U: Unknown or chaotic predicate evaluation.

We use these features in traditional classification algorithms, including 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 [4, 5].

5 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 (Equation 5, Page 400 in [4]) with respect to the true orbit classes for each individual feature. We consider two cases: one where each orbit is described by 1000 points and the other where each orbit is described by the first 150 points. Figure 7 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 7, respectively. These results suggest that the point-level density of the points is a good discriminant for the class. Also, the chaotic evaluation seems to be useful in that it indicates when many points lie along the diameter and is conceptually similar to the $f_{primary}$ feature.

6 Experimental Results
We evaluate our approach to orbit classification through two experiments. In the first experiment, we compare the accuracy of the two KAM implementations to standard feature-based classifiers from the Weka software toolkit [5]. In the second experiment, we evaluate the robustness of the classifiers when they are applied to orbits with few points.

6.1 Experimental Procedure
We used 6 multi-orbit plots each containing 100 orbits from simulations of the CDX tokamak at PPPL. The number of points in each orbit is variable. We hand labeled these orbits to

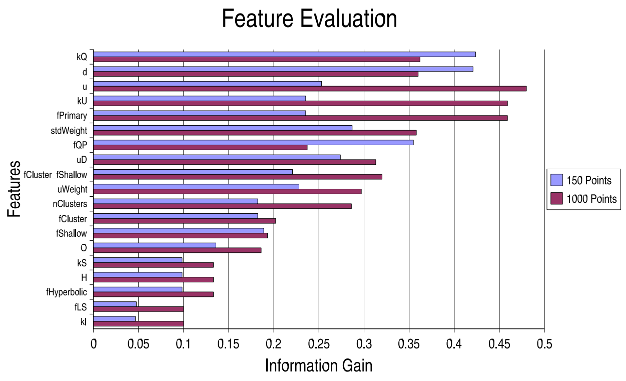

Figure 7: Comparative feature evaluation results with 150 and 1000 points. Higher values indicate better discriminative ability.

create a training set. In both experiments, for instance-based classifiers such as the nearest-neighbor algorithm, we normalized the features to be in the range $[0, 1]$.

Experiment 1: Classification Accuracy We compare 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 using the first 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.

Experiment 2: Reduced Number of Points In this experiment, we perform a similar comparison by performing 10 runs of 10-fold stratified cross-validation. The classifiers are trained on features from orbits with 2500 points, but tested on orbits with N points. This allows us to determine the accuracy 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.

6.2 Results and Discussion Tables 3 and 4 show the results from the two experiments. We indicate how well each classifier performs relative to the custom KAM classifier which was chosen instead of the default KAM classifier as it was designed to be more suitable for our data sets. Figure 8 is a graphic illustration of the results and is included for clarity. The standard deviations for the KAM classifiers 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 are identical in Tables 3 and 4.

Figure 9 shows the time to extract features for an average orbit as we increase the number of points. This indicates the tradeoffs we need to make between the higher accuracy resulting from the inclusion of 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. Comparing the results of the two experiments, we observe that the performance is similar for large numbers of points. That is, training and testing on a large number of points (Experiment 1) gives 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 (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 10. 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, they use the features generated by the graph-based approach of KAM. If KAM cannot correctly identify the 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 graph-based KAM features also explains why the feature-based algorithms perform better when trained and tested on the same number of points. Orbits with few points clearly appear different

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

Table 3: Results from Experiment 1 showing accuracy and significance with respect to the custom KAM classifier. All classifiers were trained and tested on orbits with an increasing number of points as indicated in the Dataset column.

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

Table 4: Results from Experiment 2 showing accuracy and significance with respect to the custom KAM classifier. All classifiers trained on orbits with 2500 points and tested on orbits with an increasing number of points as indicated in the Dataset column.

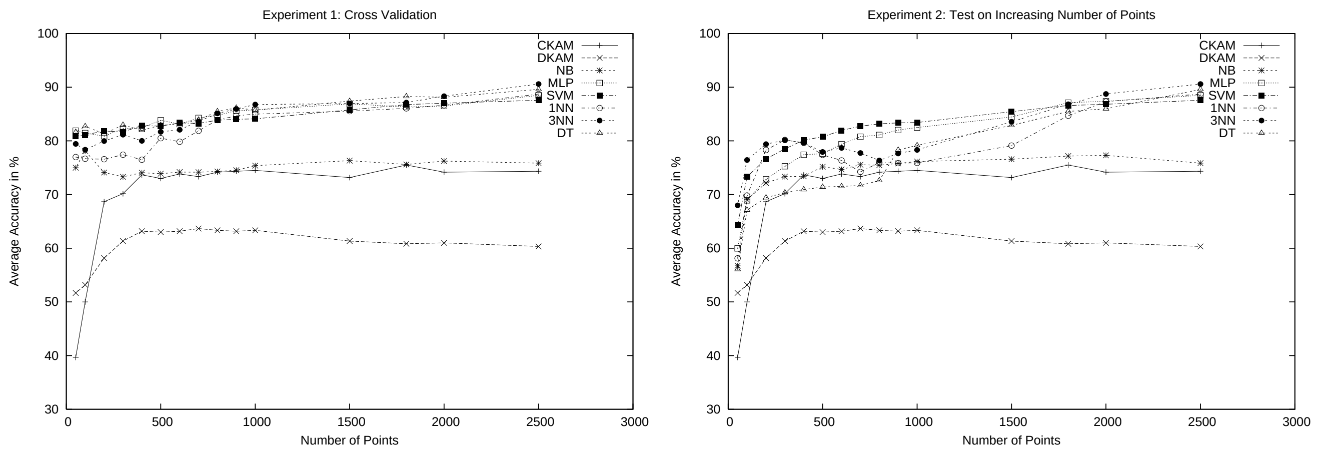

Figure 8: Comparative performance for Experiment 1 on accuracy of classifiers (left) and Experiment 2 on the effect of reduced number of points (right).

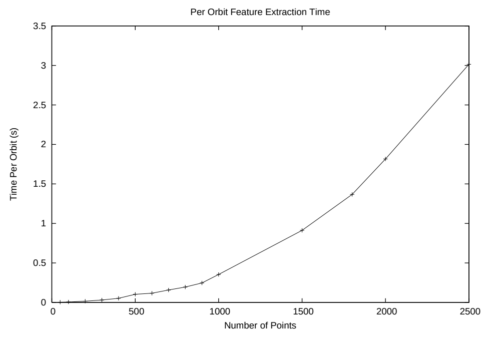

Figure 9: Per-orbit feature extraction time for orbits with an increasing number of points.

from orbits with many points. As a result, the features extracted are different. Therefore, one cannot expect a feature-based classifier trained on orbits with many points to perform well on orbits with few points.

Interestingly, the feature-based classifiers achieve relatively good performance in spite of the difficulties with few points. In the quasi-periodic orbit from the figure, the MST is visually similar with 150 or 1000 points. Although the difference is much larger for separatrices, the edge-related features such as the uniformity and edge statistics are preserved. These features have the highest informative value as shown in Figure 7. In contrast, the default KAM classifier focuses more on the intuitive but less informative features such as $f_{hyperbolic}$ and $n_{cluster}$, resulting in poorer performance.

In addition, the accuracy of the default 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 default KAM approach was designed using the Henon map as the source of the plots [7]. Figure 11 shows a cross–point region of a separatrix orbit from a Henon map and from our data set, along with the MST. In the Henon map, there is a dense collection of points near the cross-point compared to points further along the diameter. The lobes are also significantly wider than the inter–point distance along the diameter. As a result, the MST can effectively capture the structure of the orbit with as few as 150 points. In our data set, the lobes are much thinner, with the width of the lobe approaching the inter-point distance along the diameter. As a result, the MST has branches that cross the lobes. When the number of points is small, the MST can no longer capture the structure of the orbit. Further, in

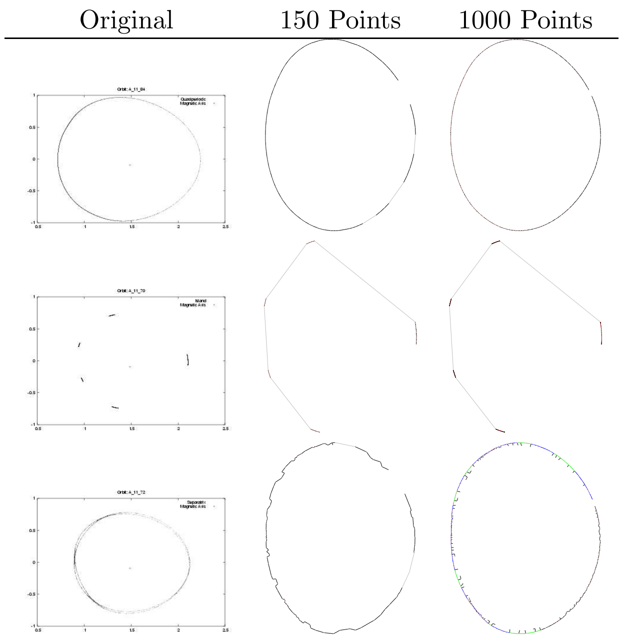

Figure 10: Orbits illustrating the effects of reduced number of points for the quasiperiodic (top), island chain (middle), and separatrix orbits (bottom).

the separatrix from the Henon map, one can identify a clear cross point, even with 150 points. In contrast, for our data, the cross points are very noisy and contain many small branches. As a result, the KAM approach can only identify cross points near their true location.

7 Related Work

The analysis of Poincaré plots, in particular, the problem of classification of orbits, has been studied both in the physics and in the machine learning communities. Depending on the information available, the approaches taken very. For example, if the equations describing the particle trajectories are known, orbits can be classified based on the parameters of the equations [3].

In our case, however, we are provided the points which make up an orbit. The approach typically used is to classify the orbits based on a description extracted from the points. In the paradigm known as spatial aggregation [8, 6], an approach based on visual reasoning is proposed. 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

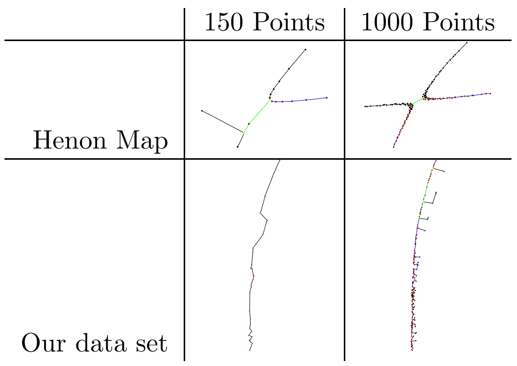

Figure 11: Zoomed-in areas of a Separatrix lobe from orbits with 50 and 1000 points, illustrating the differences in separatrices in our data and in the one from the KAM book [7]. Top row: Henon map with parameters $x = 0.5551$, $y = 0.1774$, and $\alpha = 1.3284305$. Bottom row: a separatrix orbit from our data set with thin lobes. Cross points are at the centers of the figures.

includes the analysis of Hamiltonian systems in the KAM approach [7] used in this paper and the MAPS system, which designs control laws based on a geometric analysis of the state equations of a dynamical system [8].

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.

8 Conclusion

In this paper, we consider the problem of classification of orbits from fusion devices. We implemented and extended an existing rule-based algorithm called KAM which uses features extracted from the graph representation of the orbit. We customized KAM to be more suitable for our dataset and compared the performance of the rule-based approach with several feature-based classification algorithms.

Our experimental results 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. In addition, when the number of points in an orbit is small, as is the case in experimental data, the accuracy of our approach is relatively stable. Accuracy improves when the algorithm can train and test on orbits with the same number of points.

Our conclusion is that existing classification algorithms like the k–nearest neighbor and the decision tree hold the promise of providing automated classification of orbit data. They outperform both the default and our customized KAM classifier, 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 after off-line training.

Our future work will concentrate on evaluation of more sophisticated single-orbit features as well as further testing with orbit data from different simulations.

Acknowledgments UCRL-CONF-215802: 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 PPPL. 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] A. Bagherjeiran and C. Kamath. Graph-based techniques for orbit classification: Early results. Technical Report UCRL-TR-215690, Lawrence Livermore National Laboratory, 2005.

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

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

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

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

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

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

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