Inductive Learning of Feature-Tracking Rules for Scientific Visualization

Arunava Banerjee   Haym Hirsh   Thomas Ellman
Department of Computer Science
Hill Center for the Mathematical Sciences
Busch Campus
Rutgers, The State University of New Jersey
Piscataway, New Jersey 08855
{arunava,hirsh,ellman}@cs.rutgers.edu

Abstract

Numerical simulation and scientific visualization are often used by scientists to help them understand physical phenomena. One approach taken by some visualization systems is to identify and quantify coherent features in a simulation and track their trajectories as they evolve over time. Such feature-tracking systems operate either by relying on manual (human) efforts, or by utilizing ad hoc programs embodying heuristics that are computationally expensive to use. Our research demonstrates the use of inductive learning to construct feature-tracking programs for fluid flows. Our approach uses manually generated feature trajectories as training data, and applies inductive learning to construct feature-tracking rules that can then be incorporated into a feature-tracking program. This results in a more efficient system that can match up objects across large time steps without inspecting intermediate steps. We demonstrate our approach on the problem of tracking vortices in turbulent viscous fluids.

Keywords: mechanical engineering, scientific visualization, decision tree induction.

1 Introduction

Continuing advancement in the power and speed of computers have opened up new opportunities in a wide range of engineering tasks. One example is the use of a combination of computer simulation and scientific visualization to help engineers better understand various physical phenomena. It is often the case that the mathematical models underlying such phenomena are too difficult to understand analytically, so numerical simulations of the mathematical models are studied instead. Unfortunately, such simulations typically generate datasets at too detailed a level to be understandable by humans, and thus humans rely on visualization tools that depict such datasets in helpful visual forms.

One important task often faced in such work is the need to track the trajectory of objects in a simulation as they evolve over time. For example, it might be necessary to follow the movements of clouds in meteorological data, bubbles in fluids, fissures in land masses, and more generally the progress of coherent objects in many other forms of simulation data. Unfortunately, object tracking tends to be a job either done manually, or via ad hoc programs embodying simple heuristics that can be computationally costly to use. For example, to match up objects at the start and finish of a simulation, typical *feature-tracking* programs track objects through intermediate time steps of the simulation, a process that can be quite costly since just the loading of datasets from disks is a time-consuming task.[1]

The goal of this work is to use inductive learning methods on examples of tracked objects to learn accurate and tractable feature-tracking rules. Our approach is to generate a collection of examples of pairs of objects in different time steps, each pair labeled by whether or not they correspond to the same object. We use inductive learning to form a classifier that takes as input the description of two objects in different time steps and predicts whether or not they correspond to the same object. This classifier is then used to track objects by comparing an object in one time step to all objects in the later time step to see which match up according to the learned classifier.

Our inductive learning approach can be used to construct a new feature tracking program, where none previously exists. It can also be used to improve the performance of an existing, heuristic feature-tracking program. In the first case, we use a human as the "gold standard" for learning: Given new simulation data without an appropriate tracking program, we rely on a human to label pairs of objects in different time steps of the initial portion of the simulation, according to whether or not they refer to the same object. Inductive learning generates tracking rules that can then be used for the remainder of the simulation. Our system thus uses learning to "bootstrap" the human analysis of the initial portion of the simulation into an automatic analysis of the remainder. In the second case, we use the existing tracking program as the "gold standard" for learning: The existing program is used to track objects incrementally over a long period of time, and to label pairs of objects in widely separated time steps as to whether they are the same object. Inductive learning generates rules that decide whether objects match, without looking at intervening time steps. Our system thus uses learning to "compile" the results of the existing feature-tracking program into a more efficient form.

We have applied our approach to the problem of learning rules for tracking vortices in turbulent viscous fluids. We begin with a description of this domain and the existing approach to tracking vortices. We then give details and an evaluation of our application of inductive learning in this domain. We conclude with a summary and discussion of future work.

2 Tracking Vortices in Turbulent Viscous Fluids

The goal of computational fluid dynamics (CFD) is to gain a better understanding of the dynamics of fluids through the use of computer simulations [5, 2, 7, 8]. Scientists can, in principle, gain an understanding of all such phenomena by conducting numerical simulations of the Navier-Stokes equations that govern the dynamics of fluids. Shifting from an understanding of the dynamics of

[1] In this paper we use the term "feature" to refer to coherent objects in a simulation, to be consistent with its use in the scientific visualization literature. It should not be confused with the use of the term "feature" in machine learning — we use the term "attribute" in such cases instead.

fluid particles at this low level to a comprehension of the global behavior of fluid masses under variable conditions has, however, proven to be a difficult task.

Scientists have begun using visualization tools to enhance their understanding of turbulence phenomena [3, 4, 1, 6, 11]. First, the temporal behavior of all particles in a mass of fluid under turbulent conditions is simulated using the Navier-Stokes equations. The fluid mass is then segmented into regions based on computed characteristics of constituent particles. These regions can be measured, displayed and tracked over time, enabling scientists to observe how these segmented regions evolve.

One category of abstract objects studied in CFD are *vortices*. A vortex is a closed region, every element of which has a *vorticity* larger than a specific threshold. The region corresponding to a specific vortex is computed by first calculating the *vorticity vector field*, which corresponds to the **curl** of the velocity vector field (defined by the trajectories of the fluid elements), and subsequently segmenting the complete space into closed regions, using a threshold operator on the *magnitude* of the vorticity field. Figure 1 depicts vortices that result from performing such operations on a specific case of a turbulent fluid medium.

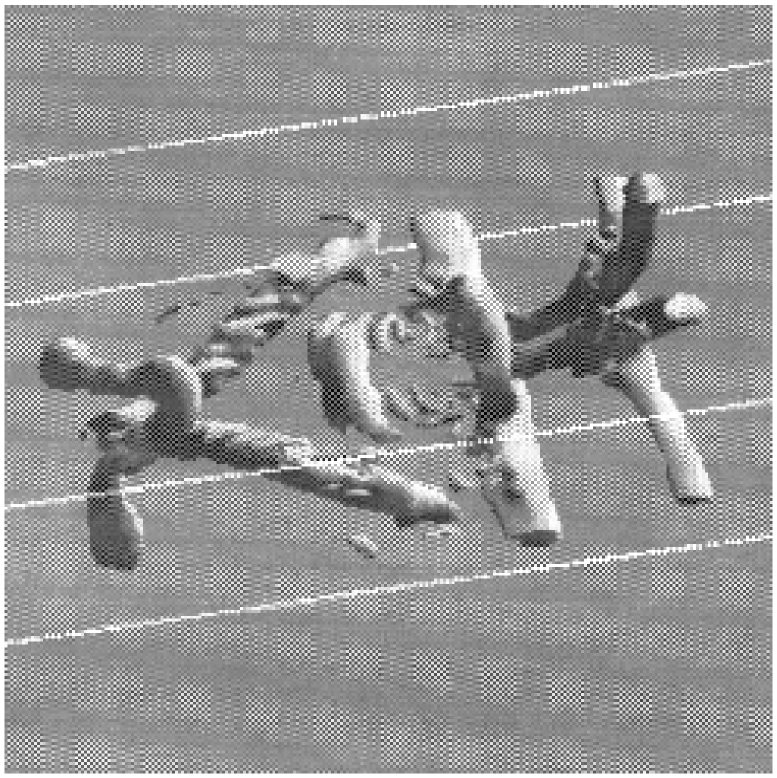

Figure 1: A snapshot of vortices in a turbulent fluid medium.

Tracking a vortex through time in a three-dimensional fluid space is a very difficult task. Not only can vortices change shape very rapidly, they can split into multiple vortices, merge with other vortices, come into existence quite unexpectedly, and cease to exist in a similar fashion. Vortices are, however, known *not* to translate very rapidly through space. Most tracking programs in this domain therefore use physical proximity as a cue to track objects. The tracking program on which our work is based matches up objects in one time step with objects in the next time step in the following fashion.

- Object O in one time step has split into objects $O_1, \ldots, O_n$ in the next time step if each of $O_1, \ldots, O_n$ are close to the location of O as well as to one another and the sum of the volumes of $O_1, \ldots, O_n$ is close to the volume of O.
- Objects $O_1, \ldots, O_n$ in one time step have merged into object O in the next time step if each of $O_1, \ldots, O_n$ are close to the location of O as well as to one another and the sum of the volumes of $O_1, \ldots, O_n$ is close to the volume of O.

Objects in one time step that are not matched up with objects in the next time step are labeled as disappearing at the given time step, and objects in the second time step that do not match up with objects in the preceding time step are labeled as emerging at the given time step. Note that the tracking program matches objects in consecutive time steps only with respect to their positions and volumes. Moreover, notions such as "close" are defined by the tracking-system creators in an ad hoc fashion to yield suitable tracking results on the given simulation.

3 Learning Feature-Tracking Rules

The preceding feature-tracking method exploits the fact that objects do not move very far in consecutive time steps. The use of this tracking program is therefore only appropriate when tracking is done over *small* time steps. The accuracy of feature-tracking degenerates as the length of the time step is increased. In order to determine whether an object at a specific time corresponds to another at a later time, where the time elapsed is large, the tracking program has to track the object incrementally through intermediate time frames. The time complexity of tracking objects across large time gaps is therefore very large.

To explore our method for learning feature-tracking rules, we applied it to learn rules for tracking objects across distant time steps without inspecting intermediate time steps. Vortices are defined as an ordered set of attributes representing

- the total volume of the vortex,
- the total mass of the vortex, defined as the summation of local vorticities over the complete volume of the vortex,
- the position of the centroid of the vortex defined with respect to a fixed set of co-ordinate axes, and
- the six second order moments of inertia of the vortex, defined with respect to the same set of co-ordinate axes.

Our training data consisted of descriptions of pairs of objects labeled by whether they correspond to the same entity. The set of attributes describing the pair of objects fall into five categories, those describing

- the vortex in the first time step,
- the environment of that vortex in its time step,
- the vortex in the second time step,
- the environment of the second vortex in its time step, and
- other global properties.

The attributes used to describe a vortex were its volume plus all of its second order moments. The attributes describing the position of the vortex were discarded because the vortices were noted to move large distances over the intervals of time across which they were matched. The attribute representing the mass of the vortex was also discarded because it yielded inferior results. The environment of a vortex was described by the volumes and the distances of five closest vortices. The global properties consisted of the time gap between the two time frames, the euclidean distance between the centroids of the two vortices, and the difference between the volumes of the two vortices. In all there were forty attributes. The concept classes were defined as a "1" if the two vortices were *related* to each other or as a "0" if not. Here, two vortices are said to be related to each other if the vortex further on in time was a result of any number of merges and splits that involved the first vortex.

The simulation dataset on which we conducted our experiments has 240 time frames, each frame containing about 10 large vortices and about 90 other smaller vortices. To mimic the situation where a human labels some initial portion of a simulation, and learning is used to form a tracking program for the remainder of the simulation, we partitioned the dataset into two groups of 120 time frames each. Training data was generated from the initial 120 time steps and testing data was generated from the disjoint second set of 120 time steps. Examples were created by choosing two time steps and one object from each of the time steps. The examples were represented using the 40 attributes described above, and labeled by whether or not the two vortices were related to each other. (Note that this process generates $O(n^2)$ examples from class "0" for every $O(n)$ examples from class "1", where n denotes the average number of vortices in a time step.) Since many learning methods perform poorly when examples are not distributed evenly among the concept classes, the set of examples generated was randomly re-sampled to obtain a set that was balanced with respect to the frequency of objects from both classes. 96,472 examples were generated for the resulting training set, and 54,841 for the test set.

Comparative studies [9, 12] have shown that, during training, decision tree algorithms run significantly faster than connectionist methods. Since one of our objectives was to reduce the time complexity for tracking, we chose C4.5 (version 6) [10] as our learning method. We ran three random trials of C4.5 with windowing turned on. Error rates were computed as the ratio of the number of misclassified pairs to the total number of pairs representing a given time gap. Figure 2 presents the results for the best of the three trials (all three trials had very similar outcomes, generating trees containing 239, 245 (for the best tree), and 261 nodes). Error rates are plotted as a function of the number of time steps that the two objects being matched are apart. Recall that data is labeled by the tracking program applied through all intermediate time steps. The results of learning are represented by the dashed curve. As a baseline, the solid curve presents the error rate of the tracking program when applied directly to the objects in the two time steps, without it tracking through intermediate time steps. These results show that learning from an initial portion of the simulation yields fairly accurate tracking for the later portions of the simulation that was not used in training. It also shows that learning compiles the results of tracking across intermediate steps to yield results superior to direct application of the tracking program when the gap between time steps is sufficiently large.

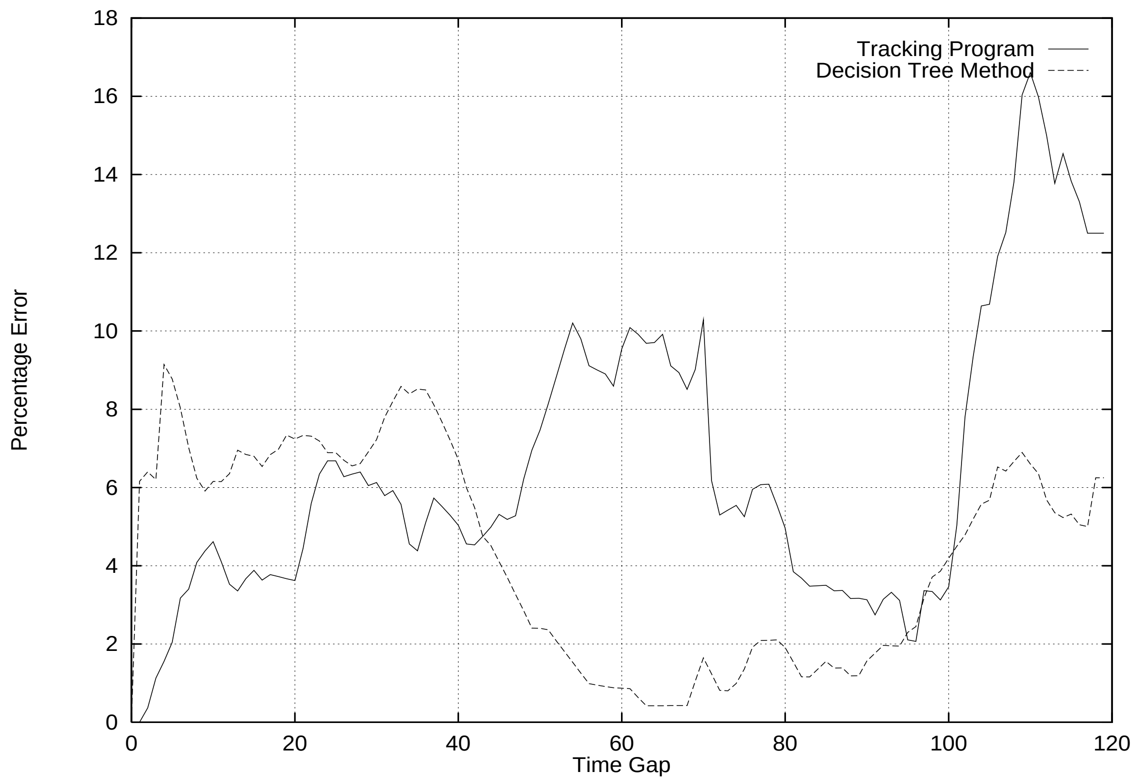

Figure 2: Training on initial 120 times steps and testing on subsequent 120 time steps.

We were surprised to discover that the accuracy of our rules improves as the distance between time steps increases - only later beginning to degenerate in quality. One possible explanation for this is that learning will only perform well if the general characteristics of two objects some distance apart in the training data are similar to the general characteristics of two objects the same distance apart in the testing data. If not, the classifier learned for objects a given distance apart in the training data would not predict as well on testing data an equal distance apart. If this explanation were correct, one would expect that the descriptions of objects in the training data that are between 40 and 100 time steps apart are similar to those objects in the testing data that are comparable distances apart. This would mean that switching the training and test sets — training on data generated from the second 120 time steps and testing on data from the first 120 time steps — should yield comparable results, with best performance for objects between 40 and 100 time steps apart. The results of this experiment are shown in Figure 3, and are consistent with this explanation.

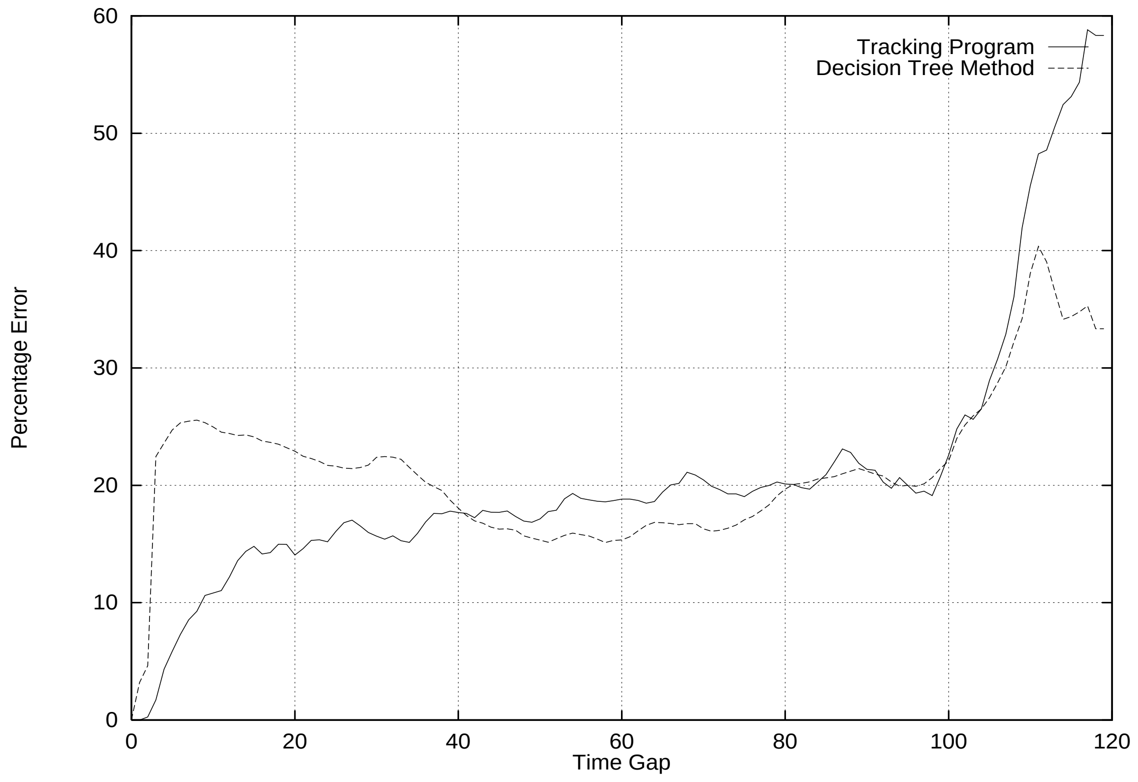

Figure 3: Testing on initial 120 times steps after training on second 120 time steps.

4 Concluding Remarks

This paper has described the application of inductive learning to learning feature-tracking rules for vortices in turbulent viscous fluids. This approach can be used to bootstrap off human tracking efforts to generate tracking programs for an entire simulation (simulated here by training on an initial portion of a simulation and testing on the remaining portion). It can also be used to compile the results of feature-tracking across intermediate time steps to enable feature tracking of objects across distant time steps without inspecting intermediate steps.

There are a number of important directions for future work. First, we would like to understand how our method behaves as a function of the amount of classification noise in the training data. Second, this paper has described the application of our method in one engineering domain; there are many other domains that entertain similar problems. Third, we have used only one learning method, C4.5, and other methods may yield different results. Fourth, we have explored our approach with one feature-tracking program. It would be valuable to explore the quality of learned feature-tracking rules when data are labeled using other feature-tracking methods. Finally, in most contexts simulation data come from some physical domain where a rich collection of background knowledge is often available. Integrating background knowledge into the learning process could prove an important source of additional information.

Acknowledgments

This research has benefited from numerous discussions with members of the Rutgers HPCD project, including Saul Amarel, Deborah Silver and Norm Zabusky. We would also like to thank Ravi Samtaney and Simon Xin Wang for helping us understand the code of various software. This research is part of the Rutgers-based HPCD (Hypercomputing and Design) project supported by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DAST 63-93-C-0064

References

[1] Bitz, F., Zabusky, N. (1990) DAVID and Visiometrics: Visualizing, Diagnosing and Quantifying Evolving Amorphous Objects. Computers in Physics. pp 603-13.

[2] Boratov, O., Pelz, R., Zabusky, N. (1990) Winding and Reconnection Mechanisms of Closely Interacting Vortex Tubes in Three Dimensions. AMS-SIAM Seminar on Vortex Dynamics and Vortex Methods.

[3] Brown, M., McCormick, B., Defanti, T. (1987) Visualization in Scientific Computing. Computer Graphics, 21(6).

[4] Buning, P., Steger, J. (1985) Graphics and Flow Visualization in Computational Fluid Dynamics. Proceedings of the 7th computational fluid dynamics conference.

[5] Buntine, J. D., Pullin D. I. (1989) Merger and Cancelation of Strained Vortices. Journal of Fluid Mechanics,205. pp 263-95.

[6] Dickinson, R., (1989) Unified approach to the Design of Visualization Software for the Analysis of Field Problems. SPIE Proceedings.

[7] Dritschel, D. (1989) Strain-Induced Vortex Stripping. Mathematical Aspects of Vortex Dynamics. SIAM,NY, pp 107-13.

[8] Melander, M., V., Zabusky, N., McWilliams, J., C. (1988) Symmetric Vortex Merger in Two Dimensions: Causes and Conditions. Journal of Fluid Mechanics,195. pp 303-40.

[9] Mooney, R., J., Shavlik, J., W., Towell, G., G., and Gove A. (1989) An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. Proceedings of the 11th IJCAI.

[10] Quinlan, J., R. (1993) C4.5: Programs for machine learning. San Francisco, CA: Morgan Kaufmann Publishers.

[11] Silver, D., Zabusky, N., (1991) 3D Visualization and Quantification of Evolving Amorphous Objects. SPIE/SPSE Conference Proceedings.

[12] Weiss, S., and Kapouleas, I. (1989) An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. Proceedings of the 11th IJCAI.