Projects in Chaotic Dynamics: Spring 2012

Elizabeth Bradley, Editor

Technical Report CU-CS 1095-12

University of Colorado
Department of Computer Science
Boulder CO 80309-0430 USA

June 2012

University of Colorado
Boulder

Table of Contents

Adding Variability to Simulated Annealing Using Chaotic Systems

Frank Di Natale
University of Colorado at Boulder

Simulated annealing (SA) is a probabilistic optimization algorithm based on the concept of annealing metals in metallurgy [3, 4]. The SA algorithm runs on what is called the cooling schedule, which is typically constructed using heuristics based on problem-specific characteristics. This paper explores the possibility of using the Lorenz and Rössler dynamical systems to add variation to the cooling schedule in order to try and provide just a basic cooling schedule that can either initially explore the solution space or find the global optimum without having to look for heuristics. The concept of using the differences between points in a single dimension was the most explored concept in this study as a way to add variability to the cooling schedule. Overall, the experiment found that using the Rössler chaotic attractor actually caused the reliability of SA to worsen rather than improve.

Keywords: Annealing, cooling schedule, chaotic, dynamics

Introduction

Within the area of optimization problems, there exist a number of problems which cannot easily be solved in fields including, but not limited to, computer design automation, imaging, physics, and biology. In 1983, Kirkpatrick *et al.* proposed a general algorithm called simulated annealing (SA) as a way to solve complex problems in a less deterministic way [3]. The algorithm makes an analogy to the annealing process used in metallurgy to combine two metals and is modeled after the statistical mechanics presented by Metroplis et al. in 1953 [4].

Kirkpatrick et al. explain that the simulated annealing approach works well with heuristics, or problem-specific characteristics. They go on to explain that there are two basic approaches to heuristics: "divide-and-conquer" and iterative improvement. Both strategies produce promising solutions when the larger problem is built upon smaller, disjoint, subproblems. As the subproblems are joined or the next iterative step is performed, proximity to the global optimum is determined by the change in energy *ΔE* of the new solution. The basic form of the algorithm is as follows [2, 3, 5]:

```
Input: Problem instance, initial temperature T
Output: A solution (can be sub-optimal)
    1. Generate the initial solution at random.
    2. while (T > 0) and (not at equilibrium) do
        a. Transform current solution into a new solution by selecting a transformation uniformly at random.
        b. if $\Delta E \leq 0$, update current state with the new state.
        c. if $\Delta E > 0$, update current state with the new state with probability, $e^{-\frac{\Delta E}{k_b T}}$ where $k_b$ is some constant.
        d. Decrease $T$ according to the annealing schedule $T_{i+1} = T\alpha$, where $\alpha$ is a ratio.
    3. Output the configuration with the lowest energy.
```

A number of heuristics have been researched for various problems [2,3], which begs the question of whether or not a general method for initial exploration (or better yet, solve) a problem without having to explore the solution space of specific problems. In order to produce the variability that's needed for the cooling schedule, we will look towards the potential benefits that dynamical systems can provide towards a general heuristic to explore the solution space of problems.

For the sake of simplicity, this paper will focus on the 2D Ising model of magnetic fields. The model is rather simple, as the magnetic field is broken down into a grid with each cell containing either a 1 or a -1. As the algorithm iterates, the magnetic cells shift towards a lower energy state. This paper will consider a 50x50 grid of cells, and assess if the use of chaotic attractors to vary the cooling schedule helps to allow the simulated annealing algorithm to more frequently capture the global optimum (as opposed to becoming stuck in local optimums). Figure 1 shows the difference between a local and global optimum with respect to the Ising model.

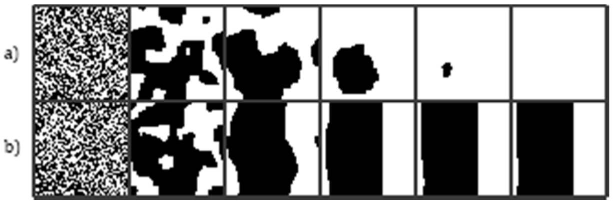


Figure 1: Time sequences of the SA algorithm on the Ising model resulting in a) global opt. and b) local opt.

Applying Chaotic Dynamics to SA

As previously stated, the goal of this paper is to explore chaotic attractors as a possible means to a general heuristic for SA cooling schedules. The idea of applying dynamical systems to SA is not an entirely novel idea. Previously, Chen and Aihara applied transient chaos to a neural networked SA approach to promote more dynamic system dynamics [1]. J. Mingjun and Tang Huanwen applied the logistic map to configuration transformations within SA in order to decrease the time needed for the algorithm to converge to the optimal solution [5]. Others have since expanded or specialized these algorithms by adding more constraints, checks for other properties, or different formulations [6, 7, 8].

The algorithm implemented in this paper remains the basic SA algorithm with the main exploration being the cooling schedule. To begin, the cooling schedule is one of the key parts of the SA algorithm which controls how quickly the system cools. Firstly, what does it mean to cool the system down? As the temperature T approaches zero, the system will accept fewer configuration changes. This characteristic means that when the system is “warmer” it is more prone to change and is more exploratory in nature. In the warmer state, the system more readily accepts a configuration change even if it does not lead the system closer to the global optimum. This state allows the system to accept potentially sub-optimal configuration in hopes that they will in the long term lead to the global optimum.

As the system cools, it becomes more set in place or is less configurable. In the cooling state the system will only accept moves that reduce the energy of the system, and therefore becomes greedier in nature. In order to vary the configurability of the system, another term is added to the temperature schedule in order to account for the variability added by the chaotic attractor. The next temperature is therefore reflected by Equation 1,

$$T_{i+1} = T\alpha(1 - x) \quad (1)$$

where x is the variability caused by the attractor. Retaining α in the equation allows for the cooling schedule to still become asymptotically smaller while still being varied so that the overall trend is still one of being cooled.

Variability Considerations

Within the scope of this paper, both the Lorenz and Rössler systems were assessed for varying the cooling of SA. The system equations are as follows with their respective constants as they were defined through this experiment:

Lorenz:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{x}\end{bmatrix} = \begin{bmatrix}a(y-x)\\ rx-y-xz\\ xy-bz\end{bmatrix} \qquad \begin{bmatrix}a\\ r\\ b\end{bmatrix} = \begin{bmatrix}16\\ 45\\ 4\end{bmatrix}$$

Rössler:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{x}\end{bmatrix} = \begin{bmatrix}-(y+z)\\ x+ay\\ b+z(x-c)\end{bmatrix} \qquad \begin{bmatrix}a\\ b\\ c\end{bmatrix} = \begin{bmatrix}.398\\ 2\\ 4\end{bmatrix}$$

The first thoughts were to map the changes in slope between data points (obtained using Runge-Kutta4) because the slope is the displacement from the last point to the current point. The x vs. z projection of both systems was arbitrarily chosen for this assessment. Appendix A contains the graphs showing the change in slope with respect to time of both systems.

The resulting slopes as pictured above have a couple of issues which do not allow them to be used as variation factors in Equation 1. Firstly, when the trajectories move in just the z direction, discontinuities appear in the graph. While the asymptotic spike is possibly favorable, initial tests resulted in the temperature rapidly increasing to infinity and never making it back down to zero. This result locked the algorithm into the exploration state into an infinite loop of accepting the new configuration. The other issue is that the slopes are otherwise extremely close to zero. Even if the discontinuities were removed, the resulting points would lock the SA algorithm into the greedy mode and force the algorithm to choose the nearest optimum (whether it were local or global).

The next thought was to take the difference between points on the same axis. This experiment worked out much better because the differences in the same direction are always finite and small enough to be able to be used as a variation factor on their own. The differences also have another advantage over the slopes in that they are much more sensitive to the step size used with Runge-Kutta4. Appendix B shows graphs for both systems with color coded plots for different timesteps. In general, to make the difference higher using a larger timestep causes a larger gap between successive points. In order to make the difference smaller, the timestep is made smaller because the difference over a smaller step is smaller.

While the difference worked much nicer than the slopes, the Lorenz attractor was completely ruled out as a possible candidate for variation. The Lorenz attractor has two basins of attraction, resulting in the difference between points to constantly shift as the trajectory loops around one attractor and then pulled around the other in the opposite direction. The difference graphs for the Lorenz system in Appendix B clearly show this fact as the difference jumps between extremes so much that the fact that the Standard SA line is an exponential is lost (as it looks almost like a straight line). The Rössler attractor, on the other hand, is much easier to change the characteristics of the differences. From experimenting with the constants in the equations, the timestep is the biggest factor in changing the variation of the differences. When the constants are changed, the patterns formed by the differences between points do change slightly, but not by a significant amount.

Experiment

The experiments to test the modifications to the SA cooling schedule were rather simple. Once the basic SA algorithm was coded and tested, it became a matter of creating functions to substitute the calculated differences in for the standard exponential cooling schedule. The only changes that were made to the SA algorithm were to let the Runge-Kutta4 algorithm run for five million steps (to be sure there were enough points for the whole annealing process) and then iterate through normally. The initial conditions for any experiments using the attractor were set to $[x, y, z] = [1.0, 2.0, 3.0]$ with timesteps of .0075, .01, and .05 tested. Each variation of the SA algorithm was run 1000 iterations, with the initial configuration randomized each iteration and all bookkeeping reset. A stable configuration was considered to be all points to be the same color (black or white), but with a threshold of 2% of the points allowed to be the minority color. This threshold was put in to account for the fact that the SA algorithm randomly permutes the current configuration which makes it statistically more difficult to permute a pixel of the opposite color. Table 1 shows the results of the experiment.

Test	Global Found	Percentage
Standard SA	687	68.7%
h = .0075	630	63.0%
h = .01	670	67.0%
h = .05	638	63.8%

Table 1: Simulated annealing test results over 1000 iterations.

As Table 1 shows, the addition of the chaotic cooling schedule appears to have made it more difficult for the SA algorithm to find the global optimum. At first glance, this result may not make sense but after looking through the algorithm's mathematical foundations it is reasonable. The temperature controls to what degree the configuration is set. Using the chaotic variation causes the SA algorithm to accept points that may have been properly set. If enough of these points are then set incorrectly by the fluctuating temperature, it may just be enough to slide the algorithm to a local optimum rather than the global optimum.

Conclusion

With the analysis of Table 1, it can be safely concluded that in the difference between points is not an effective way of adding variation to the cooling schedule of the simulated annealing. The percentage of times that the chaotic cooling schedule found the global optimum was at best 1.8% behind the general SA algorithm, and in worst cases a terrible 5.7% off. In no cases was there an improvement over the general algorithm.

Future Work

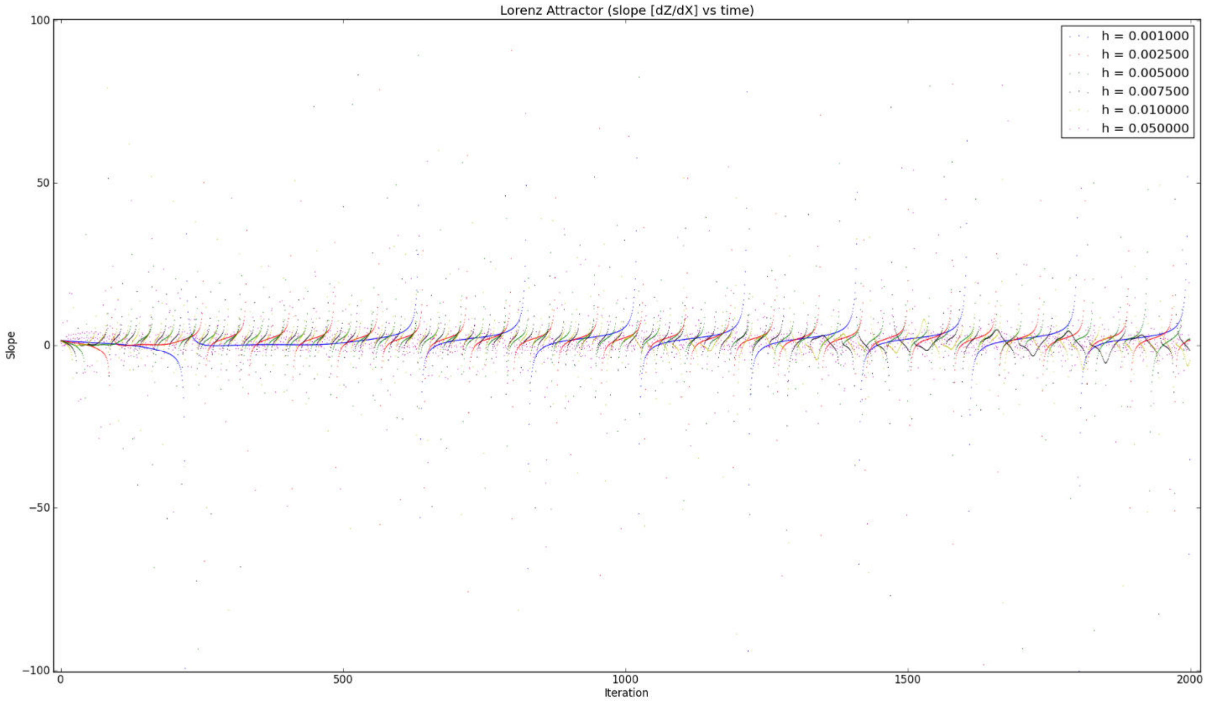
This experiment established that the difference between points is clearly not an effective way to improve simulated annealing. Other thoughts that may be potential concepts to try in the future are using the chaotic attractor as a way to set stages for the schedule. In this respect, maybe the potential of multiple annealing schedules at different points in the iterative process can be used. A simpler extension would be to try the Logistic Map with the difference method and see if that yields results because it does not have a fixed point chaotic attractor. Using the Logistic Map would be a good way to test and see if using an attractor with some form of elliptical orbit causes the effects described above of the algorithm rejecting and accepting when it shouldn't.

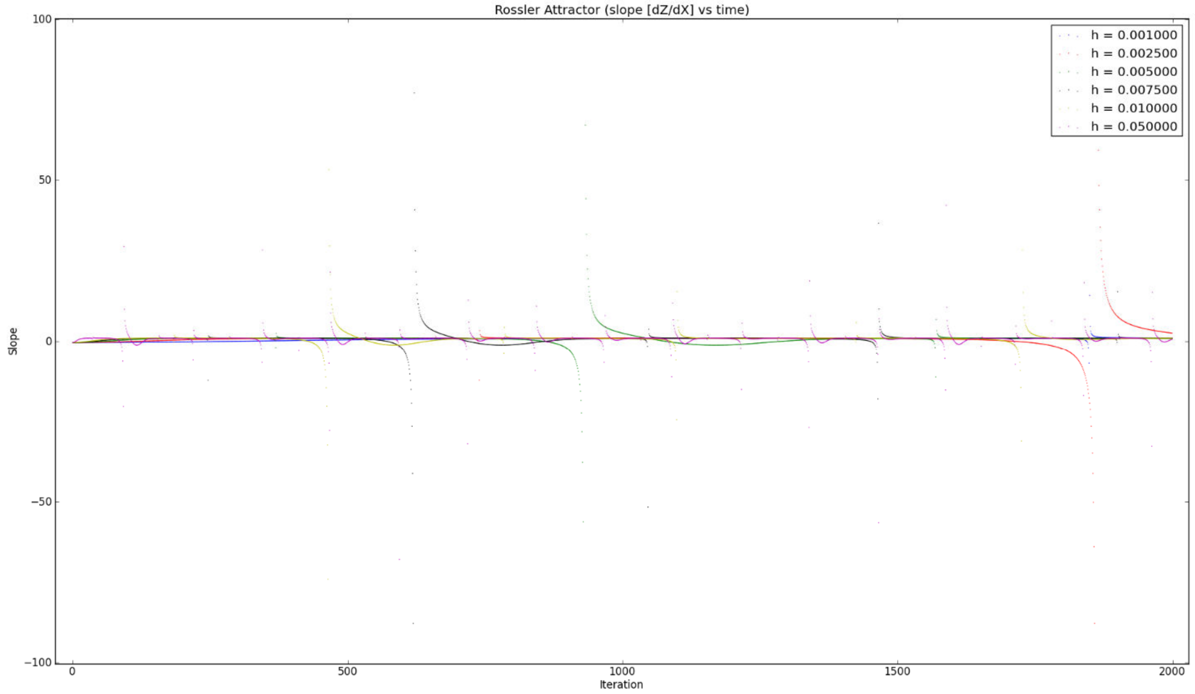
References

1. Chen, Luonan, and Kazuyuki Aihara. "Chaotic Simulated Annealing by a Neural Network Model with Transient Chaos." *Neural Networks* 8.6 (1995): 915–930. Web. 4 May 2012.
2. Ingber, L. "Simulated Annealing: Practice Versus Theory." *Mathematical and Computer Modelling* 18.11 (1993): 29–57. Web. 25 Apr. 2012.
3. Kirkpatrick, S., C. D Gelatt, and M. P Vecchi. "Optimization by Simulated Annealing." *Science* 220.4598 (1983): 671–680. Web. 24 Apr. 2012.
4. Metropolis, Nicholas et al. "Equation of State Calculations by Fast Computing Machines." *The Journal of Chemical Physics* 21.6 (1953): 1087–1092. Web. 4 May 2012.
5. Mingjun, Ji, and Tang Huanwen. "Application of Chaos in Simulated Annealing." *Chaos, Solitons & Fractals* 21.4 (2004): 933–941. Web. 24 Apr. 2012.

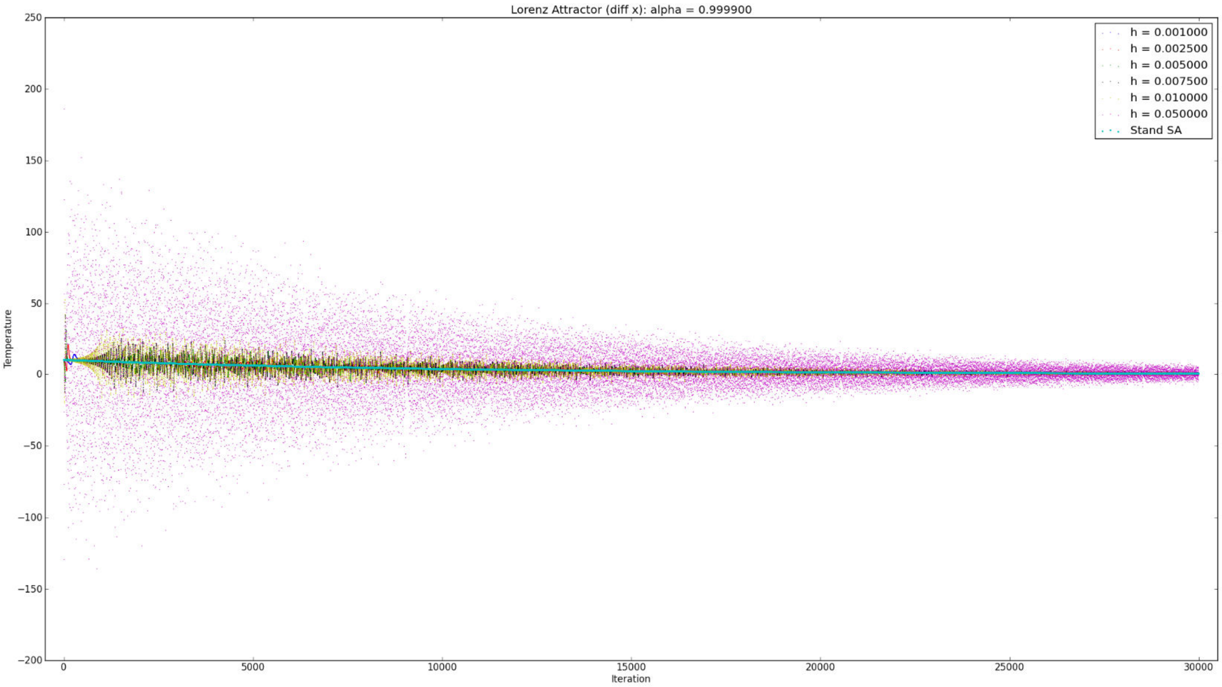
6. Wang, L., and K. Smith. “On Chaotic Simulated Annealing.” *IEEE Transactions on Neural Networks* 9.4 (1998): 716–718. Web. 24 Apr. 2012.
7. Yang, Dixiong, Gang Li, and Gengdong Cheng. “On the Efficiency of Chaos Optimization Algorithms for Global Optimization.” *Chaos, Solitons & Fractals* 34.4 (2007): 1366–1375. Web. 24 Apr. 2012.
8. Zheng, Liying, Kejun Wang, and Kai Tian. “An Approach to Improve Wang–Smith Chaotic Simulated Annealing.” *International Journal of Neural Systems* 12.5 (2002): 363. Print.

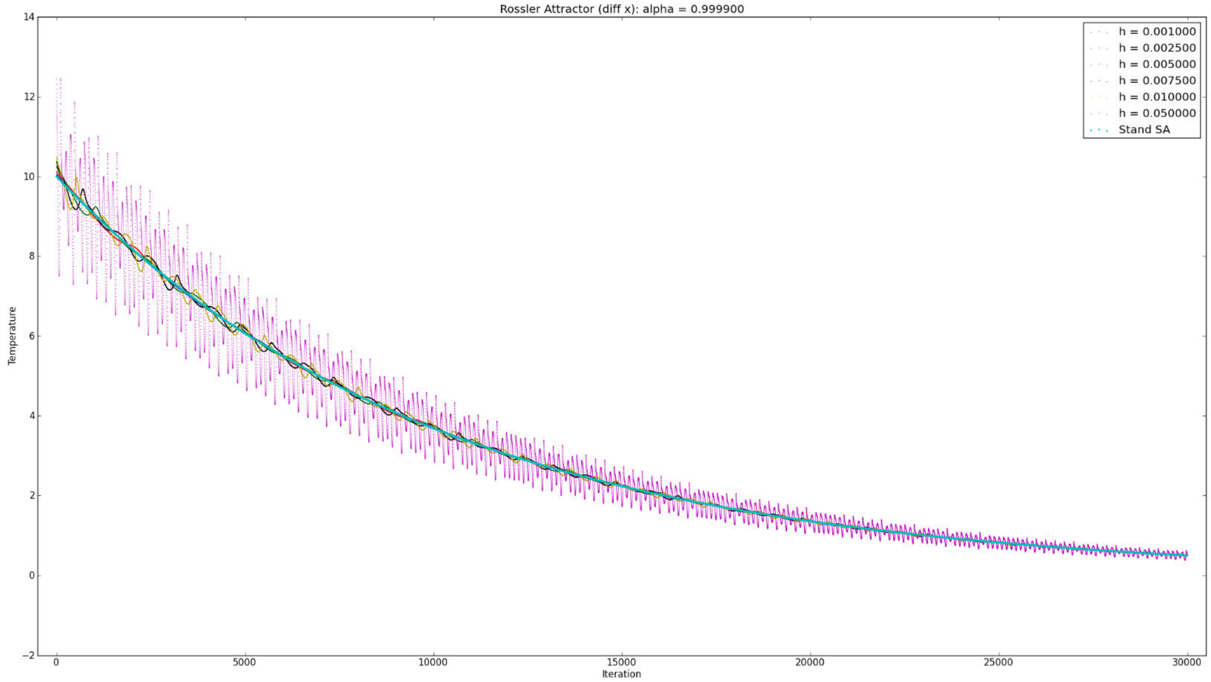
Appendix A – Slope vs. Time Graphs

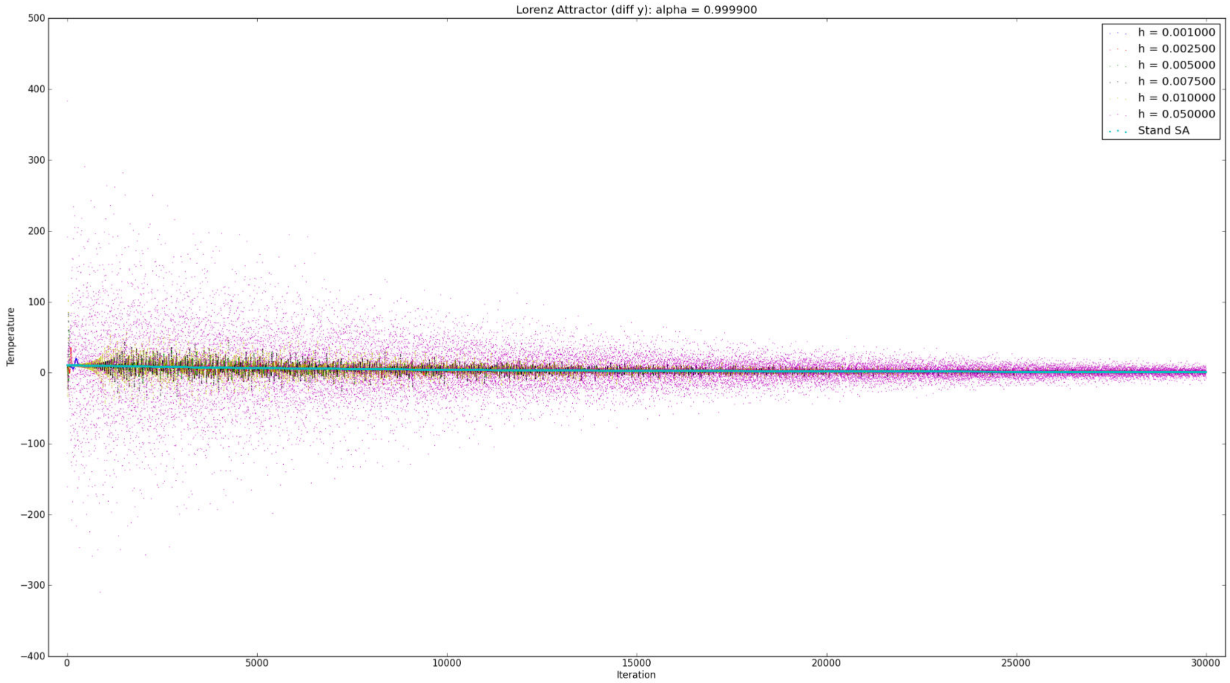


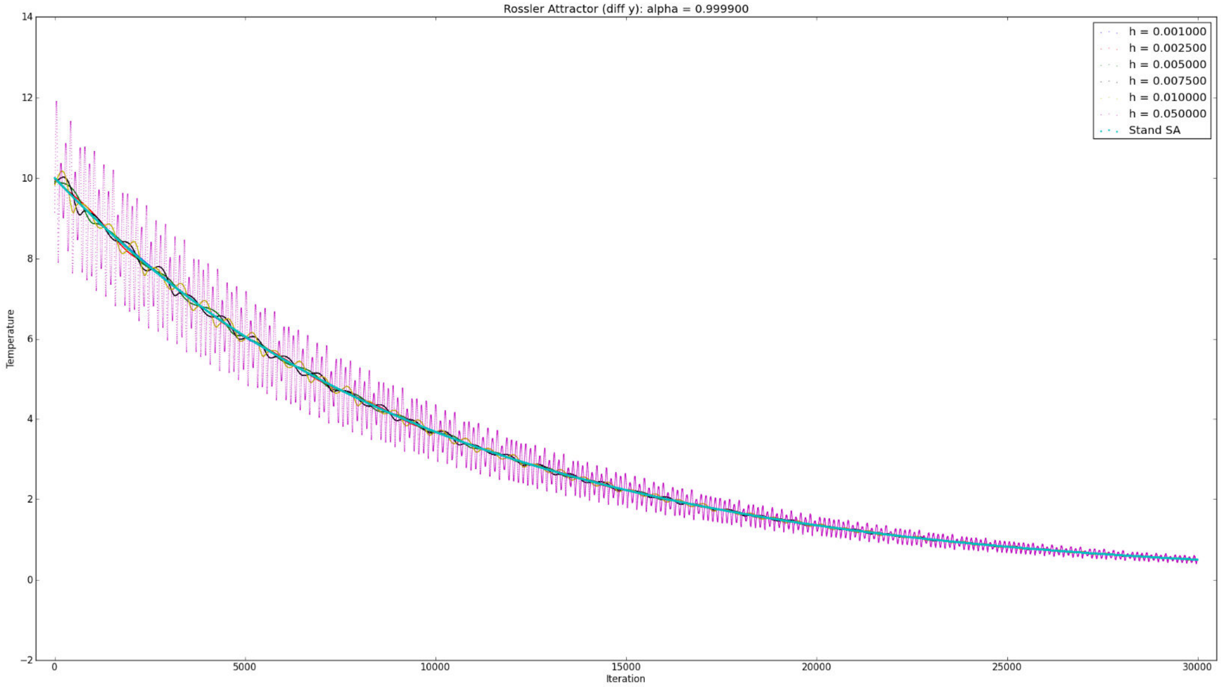


Appendix B – Difference Graphs



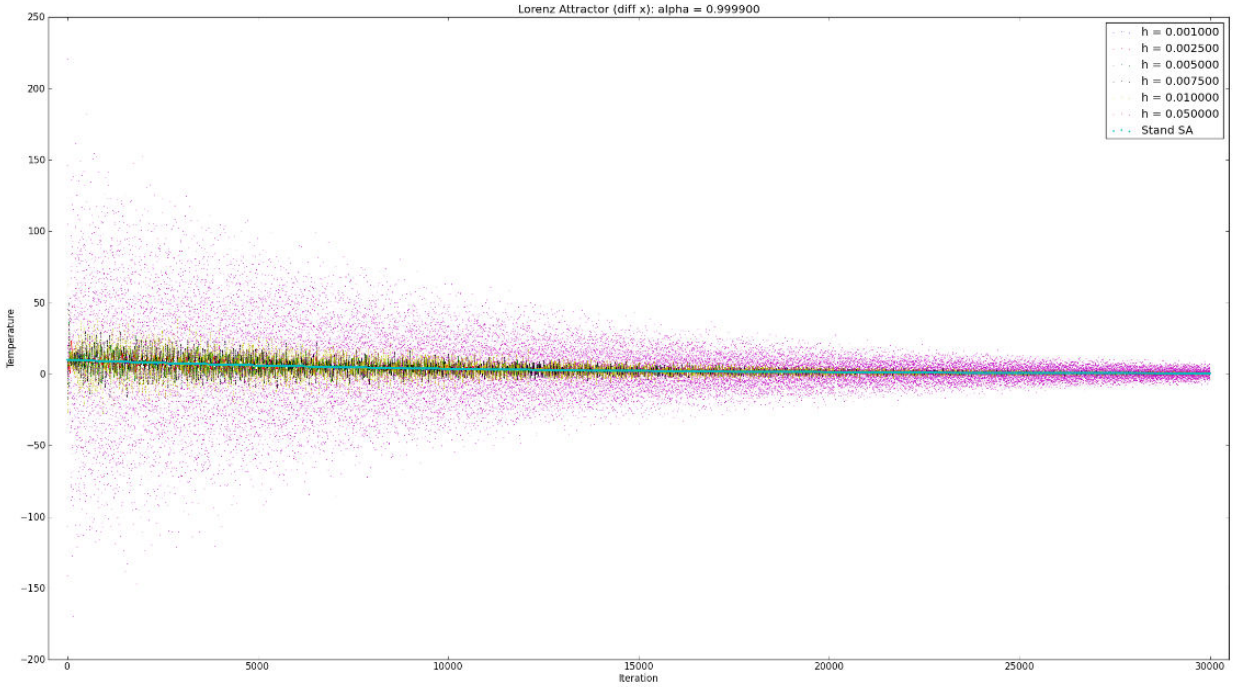


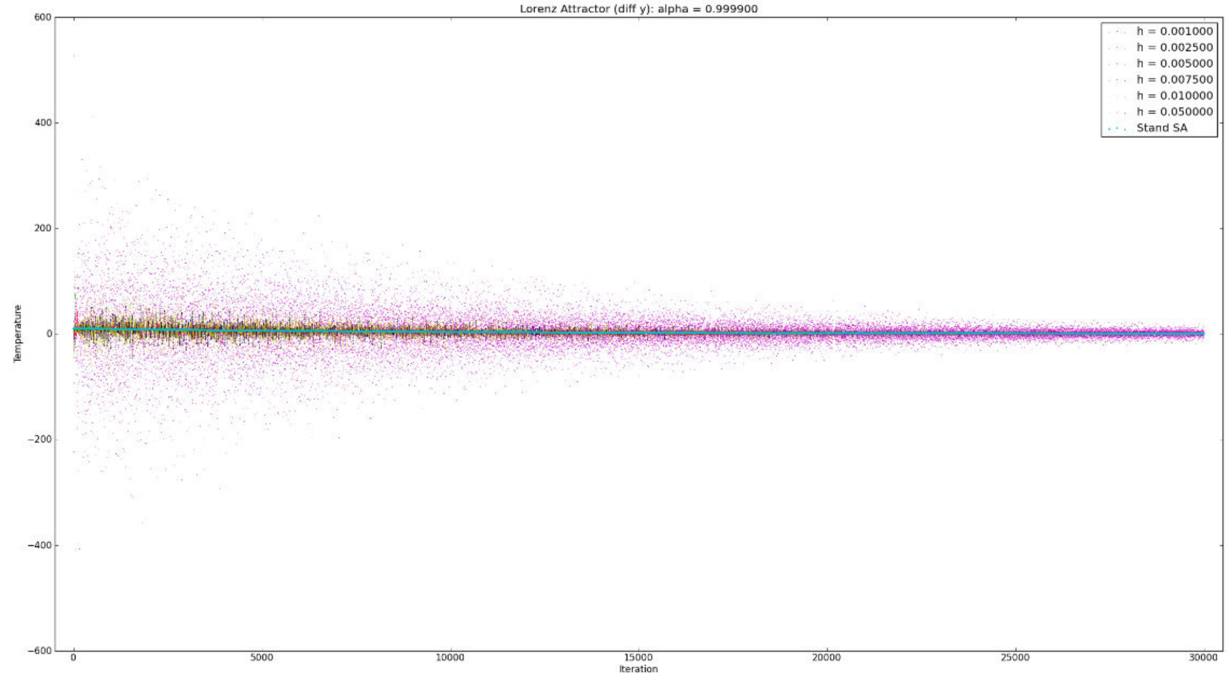

Lorenz Attractor (diff y): alpha = 0.999900
Temperature
Iteration
500
400
300
200
100
0
−100
−200
−300
−400
0
5000
10000
15000
20000
25000
30000
h = 0.001000
h = 0.002500
h = 0.005000
h = 0.007500
h = 0.010000
h = 0.050000
Stand SA


Rossler Attractor (diff y): alpha = 0.999900
Temperature
Iteration
14
12
10
8
6
4
2
0
−2
0
5000
10000
15000
20000
25000
30000
h = 0.001000
h = 0.002500
h = 0.005000
h = 0.007500
h = 0.010000
h = 0.050000
Stand SA

Appendix C – Difference Graphs for Alternative Constants (Lorenz)

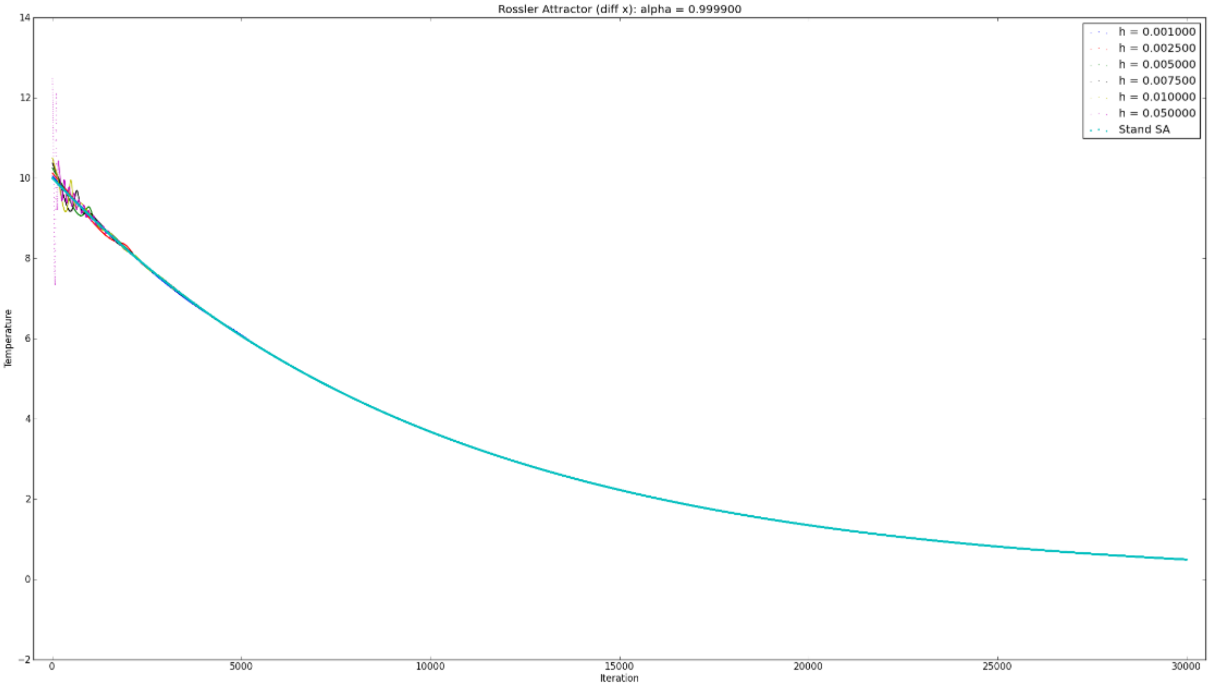
$$\begin{bmatrix} a \\ r \\ b \end{bmatrix} = \begin{bmatrix} 16 \\ 52 \\ 4 \end{bmatrix}$$

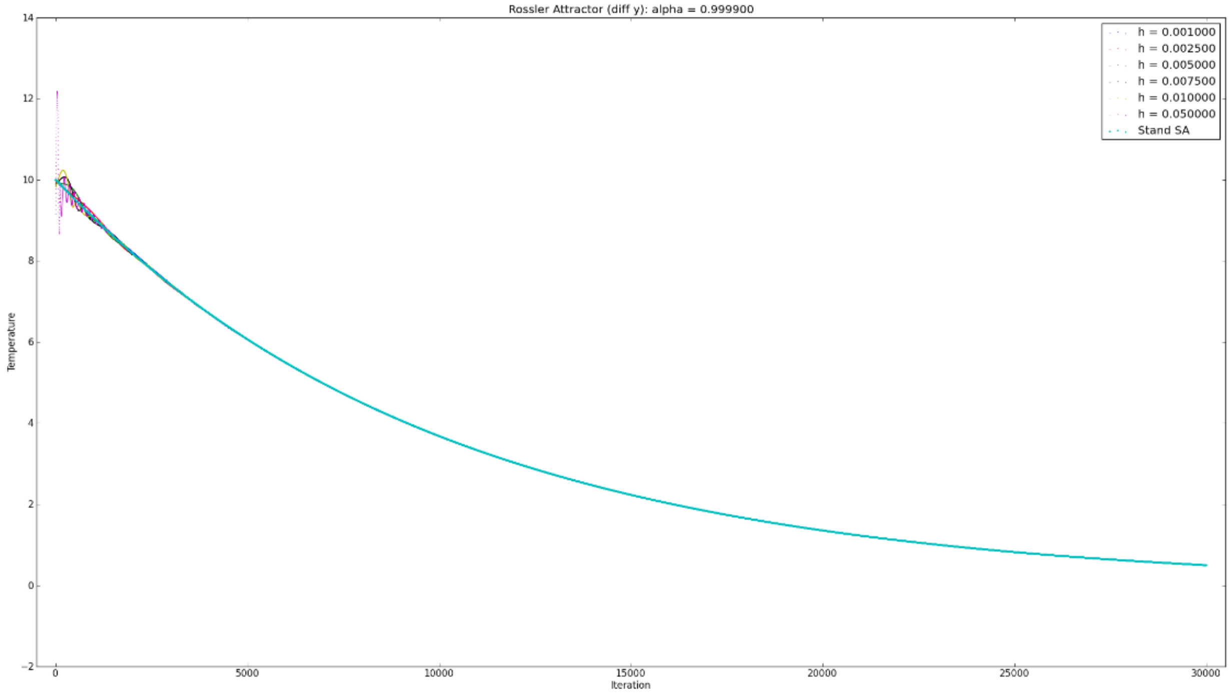




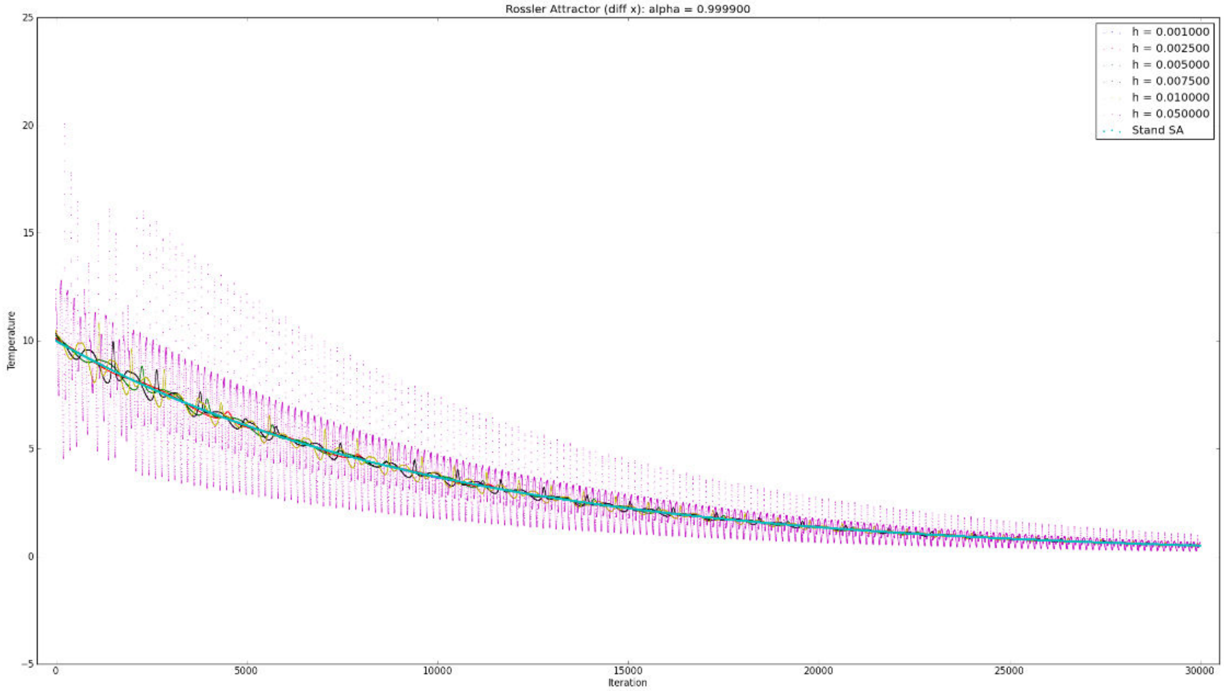
Appendix D - Difference Graphs for Alternative Constants (Rössler)

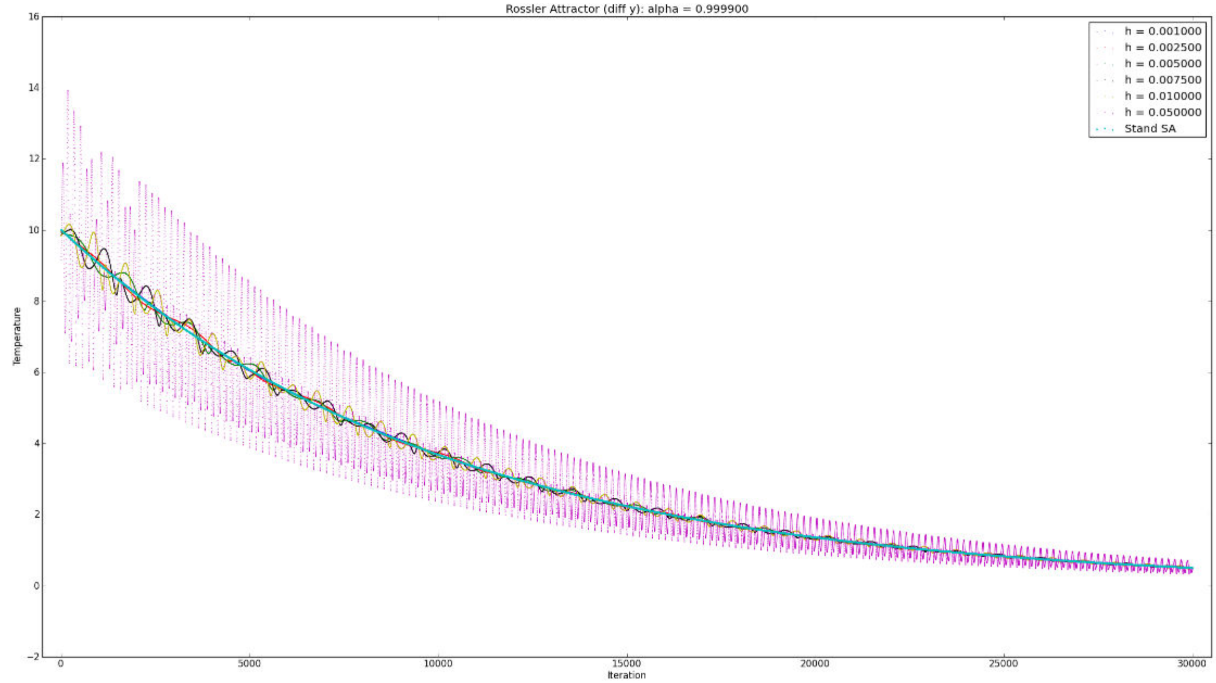
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 16 \\ 5 \\ 4 \end{bmatrix}$$





$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 16 \\ 2 \\ 4 \end{bmatrix}$$





Simulating the effects of pace cars

Robert Frohardt
CSCI 5446
University of Colorado at Boulder
Department of Computer Science
Robert.Frohardt@colorado.edu

May 6, 2012

Abstract

I implemented a traffic simulator according to the *car-following* model of traffic flow and used it to explore the effects of pace cars on highway traffic. I found that this type of model is not ideal for studying the impact of pace cars. However, to the extent that the model is valid, the simulations did suggest a safety benefit of pace cars due to the breaking of traffic into smaller groups of cars, decoupling one group from the next. Specifically, I found that cars farther back in a group of coupled cars have wider velocity distributions. Attempts to quantify this effect in terms of Lyapunov exponents were inconclusive.

1 Introduction

During the 2011–2012 ski season, the Colorado Department of Transportation (CDOT) used "pace cars" to control the flow of traffic from Summit County to Denver along I-70 during peak ski traffic. The primary objective of the Rolling Speed Harmonization program (as CDOT calls it) is to improve travel safety, although CDOT also claims that it can improve traffic flow[8]. The program operates by arranging for police vehicles to cap the speeds of groups of vehicles between 35 mph and 55 mph. CDOT has provided data on its website suggesting that the pace cars may have a positive effect on traffic flow, but the data is far too limited to draw any conclusions. The data is shown in Table 1 and pairs equivalent days from the 2010–2011 and 2011–2012 ski seasons (i.e., closest date that was the same day of the week). The dates in 2010–2011 had no pace cars, and the dates in 2011–2012 had pace cars. Travel times and speeds are for 4 pm, the peak travel time.

My goal with this project was to use traffic modeling and dynamical systems techniques to analyze the effect of pace cars on the dynamics of traffic.

2010–2011			2011–2012		
Traffic Volume	Travel Time	Speed	Traffic Volume	Travel Time	Speed
20,065	78 min	36 mph	21,349	46 min	61 mph
18,491	87 min	32 mph	22,075	65 min	43 mph
26,875	77 min	37 mph	27,794	76 min	37 mph
31,295	85 min	33 mph	30,772	108 min	26 mph

Table 1: CDOT data on pace cars

2 Traffic Modeling

There is a long history of mathematical traffic modeling. An introduction is the survey by Bellomo and Dogbe [1]. Bellomo and Dogbe describe three general strategies for modeling traffic:

1. **Microscopic**. Analogous to Newtonian mechanics. The position and velocity of the individual vehicles are modeled by a system of ordinary differential equations (ODEs).

2. **Macroscopic**. Analogous to fluid mechanics. The mass density and linear momentum of the flow are modeled by partial differential equations (PDEs).

3. **Statistical**. Analogous to statistical mechanics. The distribution of positions and velocities is modeled similarly to the Bolzmann equation.

For this project I focused on microscopic modeling. A well-studied group of microscopic traffic models are *car-following* models. These models describe the acceleration of each vehicle as some function of the state of one or more vehicles ahead of it. The first such model was published by Chandler et al. in 1958[4].

2.1 Car-following

It takes an entire survey paper to cover the history and many variations of car-following models[3]. This paper only gives a very brief overview of the theory. The basis of car-following models is the system of differential equations[4]:

$$\dot{x}_n(t+\tau) = v_n(t) \tag{1}$$

$$\dot{v}_n(t+\tau) = \alpha(v_{n-1}(t) - v_n(t)) \tag{2}$$

Where $x_n(t)$ is the position of the nth car at time t, $v_n(t)$ is the velocity of the nth car at time t, and α is a parameter of the model. In this simple version, the motion is constrained to be one-dimensional. Note that these are not ordinary differential equations (ODEs) but delay differential equations (DDEs). The implications of this are discussed in section 2.3. This system of equations does not specify $x_1(t)$ and $v_1(t)$, which describe the motion of the lead vehicle. This is addressed in section 4.1.

The physical interpretation of equation (2) is that each car attempts to match the velocity of the car infront of it. If this velocity difference is positive, car n will accelarate and if it is negative, car n will decelerate. This linear system of equations can be solved analytically for certain plausible forms of $v_1(t)$.

2.2 Variations

The next variation on the car-following equation is from Gazis et. al in 1959[5]. This version introduces a dependence on the separation between cars:

$$\dot{v}_n(t+\tau) = \alpha \frac{v_{n-1}(t) - v_n(t)}{(x_{n-1}(t) - x_n(t))^m} \tag{3}$$

The logic behind this modification is that the response of a vehicle to the velocity fluctuations of the vehicle it is following will increase as the distance between the vehicles decreases. In the case of traffic, this distance $x_{n-1}(t) - x_n(t)$ is frequently referred to as the *headway*. If the headway is large, a vehicle will only slow down slightly when it is catching up with the next vehicle. However, if the headway is small and a vehicle is still gaining on the next vehicle, it will brake as hard as possible to avoid an accident. In fact, if a vehicle is gaining on the next vehicle (i.e., $v_{n-1}(t) - v_n(t) < 0$) the acceleration approaches negative infinity as the headway approaches zero, making collisions impossible if $\tau = 0$.

2.3 Delay Differential Equations

It is important to note that in equations (2) and (3), the time-derivatives $\dot{x}_n(t)$ and $\dot{v}_n(t)$ are not given as functions of the current state of the system. Instead they are functions of the state of the system τ seconds in the past. In the context of traffic, this is an attempt to model the reaction time of drivers.

Mathematically, this makes the equation a system of *delay differential equations (DDEs)*. This is of note for (at least) two reasons:

1. As an ODE (i.e., $\tau = 0$), equation (3) does not allow collisions, but for any value $\tau > 0$, collisions are possible for certain initial conditions. Collisions do occur in real traffic flows, so in some sense, this is a desirable aspect of the model.

2. Even very simple non-linear DDEs can exhibit chaotic behavior. For example, consider the equation:

   $$\dot{x}(t+\tau) = \sin x(t) \tag{4}$$

   Sprott demonstrates that equation (4) goes through a series of bifurcations as τ increases and exhibits chaotic behavior for most values $\tau > 5$[10].

Modeling traffic as a DDE also potentially simplifies a computer simulation. This is because integration methods such as the Euler method inherently calculate based on the state of the system some discrete time-step in the past. Sprott makes this observation in the context of equation (4)[10].

2.4 Limitations

It is easy to see many limitations of this simplified model, and subsequent variations of car-following models have attempted to address some of these issues. Just a few examples of these limitations are:

- The model is one-dimensional and does not account for intersections or lane changes.
- The model only allows each car to react to the state of the next car, whereas most drivers try to see farther ahead and react to the state of all visible cars.
- The model gives all cars the same governing equations, whereas real drivers have different preferred speeds and following distances, and real cars have different acceleration and deceleration capabilities.

Invidividual studies have been able to fit equation (3) to empirical data with reasonable accuracy. Unfortunately, the calibrated values of m and α have varied widely between studies[3]. The models are at least plausible for studying general behavior of traffic, since the models seem to be good at demonstrating qualitatively “traffic-like” behavior. However, they do not have strong predictive power for any specific, real situation.

3 Implementation

3.1 Equations

For this project, I chose a further modification of equation (3):

$$\dot{v}_n(t+\tau) = \alpha \frac{v_{n-1}(t) - v_n(t)}{x_{n-1}(t) - x_n(t)} + k(x_{n-1}(t) - x_n(t)) \tag{5}$$

The term $k(x_{n-1}(t) - x_n(t))$ was first introduced in 1963[2]. I chose this model because it is relatively simple, but had been previously demonstrated to exhibit chaotic behavior[7].

The simulation uses a sine function to determine the behavior of the lead vehicle:

$$\dot{v}_1(t) = V + A\sin(\omega t) \tag{6}$$

This idea was part of the original paper on car-following[4], and was also used in the study that found chaotic behavior[7]. Some type of periodic behavior is a reasonable approximation of a car attempting to maintain its velocity by pressing the accelerator when below the target velocity, V, and releasing the accelator when above the target velocity. This has advantages over other options:

- Constant velocity: I ran simulations with this and was not able to observe any interesting behavior.

- Constant velocity modulated by random noise: Because I was looking for chaos, I did not want to confuse the issue with random noise. In the actual simulations, the simple periodic behavior of the lead car is also easy to see in the cars farther down the line. This helps give an intuitive picture of which fluctuations are directly due to the driving behavior of the lead car, and which fluctuations are potential chaotic behavior.

Another interesting possiblity would be to use the geography of I-70 or another specific road to provide the fluctuations in velocity of the lead car.

At the start of the simulation, the cars are evenly spaced at distance d_0 and all start with the same velocity V_0 (both of which are input parameters to the simulation).

3.2 Modifications

In order to get plausible simulations, I had to make a few more adjustments:

- Do not allow any vehicle to exceed maximum velocity $V_{\max}$. This turned out not to be very important as long as there was a maximum acceleration.
- Do not allow any vehicle to exceed maximum acceleration of $3.3ms^{-2}$. This was calculated to correspond to constant acceleration from $0ms^{-1}$ to $33ms^{-1}$ in $10s$.
- Do not allow any vehicle to go below a minimum acceleration of $-6.6ms^{-2}$. This was calculated to correspond to constant deceleration from $33ms^{-1}$ to $0ms^{-1}$ in $5s$.
- If any vehicle comes within $0.1m$ of the next vehicle, set its velocity to $0ms^{-1}$.

These adjustments add cusps and even discontinuities to the accelaration and velocity, which is non-physical, but they improve the stability of long simulations.

3.3 Parameters

The model is very sensitive to parameter choices. This section explores the qualitiative effects of some of the parameters. I first found a set of parameters which gave the "interesting" behavior I was looking for, then varied the parameters one at a time from this baseline to see the effect. The baseline parameters are given in Table 2. All simulations used a step-size of $0.1s$. I tested other step sizes and they had no noticable effect, possibly due to the observation about simulating DDEs made in Section 2.3. Note that the velocity $33ms^{-1}$ was chosen to correspond roughly to a typical highway velocity of 75 mph and that $20ms^{-1}$ was chosen to correspond to a pace car velocity around 45 mph.

Figure 1 shows a plot of headway versus time for the second car (i.e., the car immediately behind the lead car). This trajectory is discussed more in

α	velocity control sensitivity	$10ms^{-1}$
k	propensity to catch up to next vehicle	$0.0008s^{-2}$
τ	reaction time	$1s$
d_0	initial headway	$20m$
V_0	initial velocity	$33ms^{-1}$
V	lead-car target velocity	$33ms^{-1}$
A	amplitude of lead-car sin wave	$1ms^{-1}$
ω	frequency of lead-car sin wave	$0.05\text{rad}s^{-1}$

Table 2: Baseline parameters

Section 4. Figure 2 shows the effect of setting $k = 0$. This appears to remove the "interesting" behavior. Figure 3 shows the effect of setting $\tau = 0.1$. This also appears to remove the "interesting" behavior. Figure 4 shows the effect of setting $V_0 = V = 20ms^{-1}$. This appears to have no qualitative effect on the behavior. This is discussed further in Section 4.1. Previous papers have found that varying α led to the onset of chaos[7]. I was not able to recreate this finding.

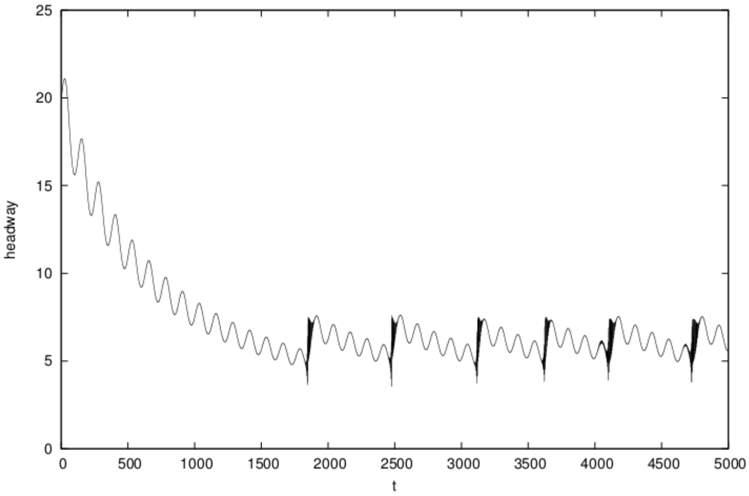


Figure 1: Baseline trajectory for car 2

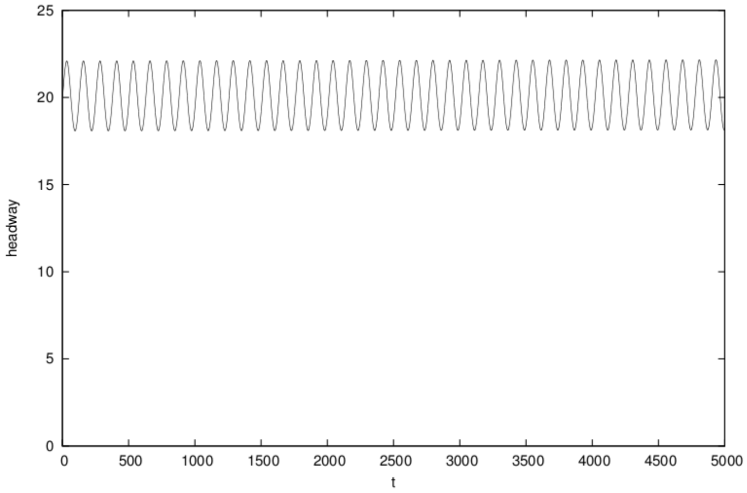


Figure 2: Trajectory for car 2 with $k = 0$

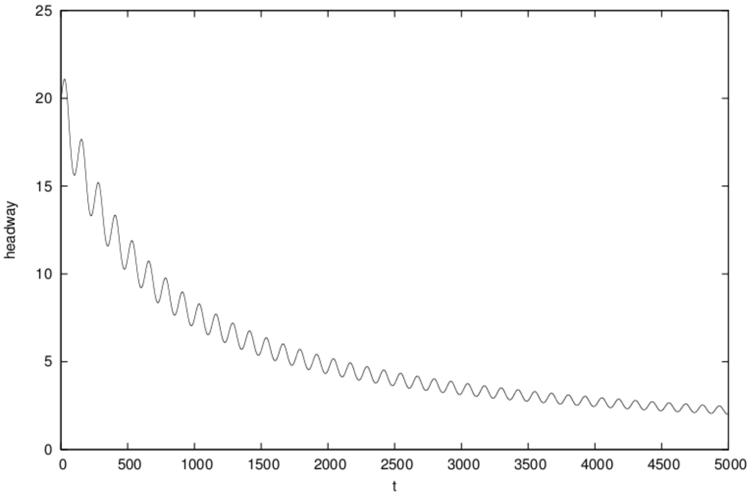


Figure 3: Trajectory for car 2 with $\tau = 0.1s$

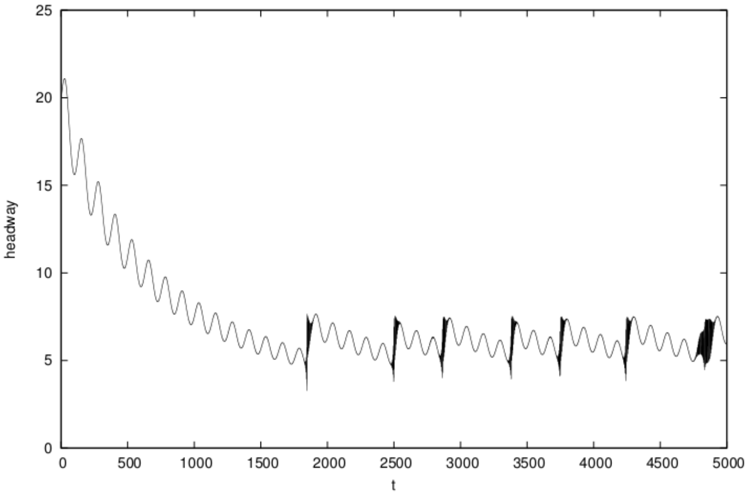


Figure 4: Trajectory for car 2 with $V_0 = V = 20ms^{-1}$

4 Results

In retrospect, car-following models are not the best models to study a change in dynamics in the presence of a pace car, because the model effectively has a built-in pace car in the lead car. This prevents analyzing the 'pace car' dynamics and comparing them to the 'no pace car' dynamics. Nonetheless, I was able to make some interesting observations relating to the effect of the pace car, although attempts to quantify these effects were unsuccessful.

4.1 Lead car velocity

As can be seen in Figure 1 and Figure 4, there does not appear to be any large effect on the dynamics due to varying the lead car velocity. This is even more evident in the comparison of state space trajectories in Figure 5.

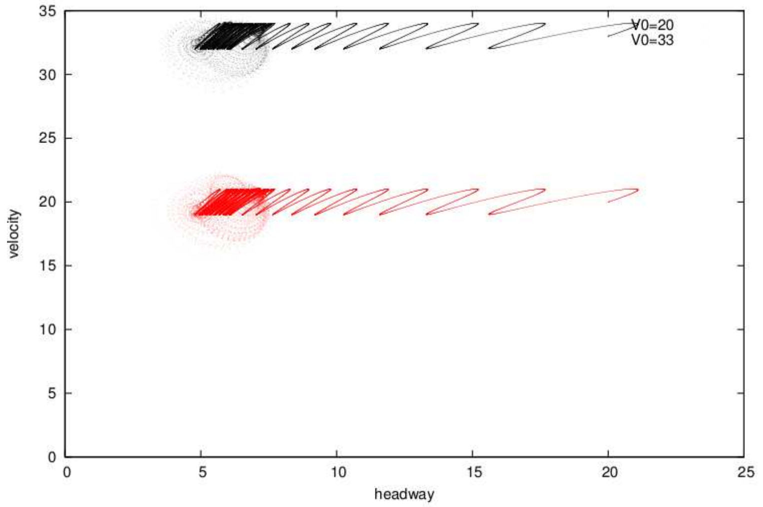


Figure 5: State space trajectory for car 2 with $V_0 = 33ms^{-1}$ versus $V_0 = 20ms^{-1}$

4.2 Attractor

The state space trajectory of car 2 has an interesting attractor. I experimented with a wide range of initial values for headway d_0 and velocity V_0, and all initial values reach the attractor quickly. The attractor is shown in Figure 6. Figure 7 shows how the structure of the attractor arises out of the periodic behavior with a series of temporal Poincaré sections at fractions of the lead car's period.

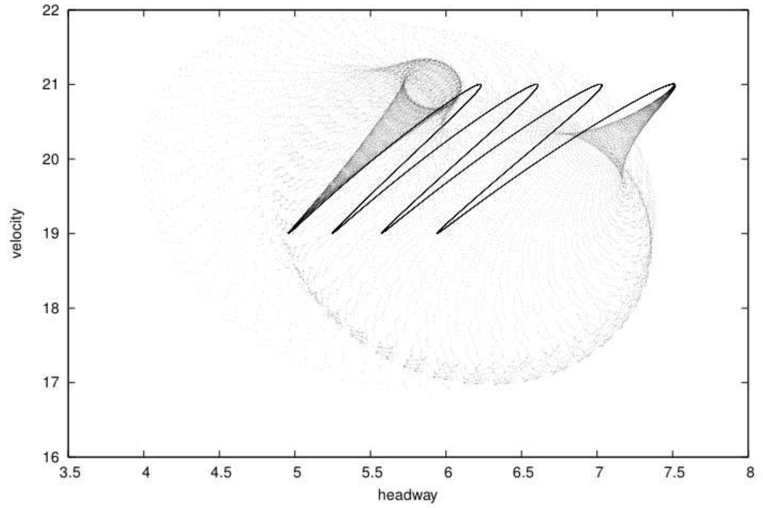
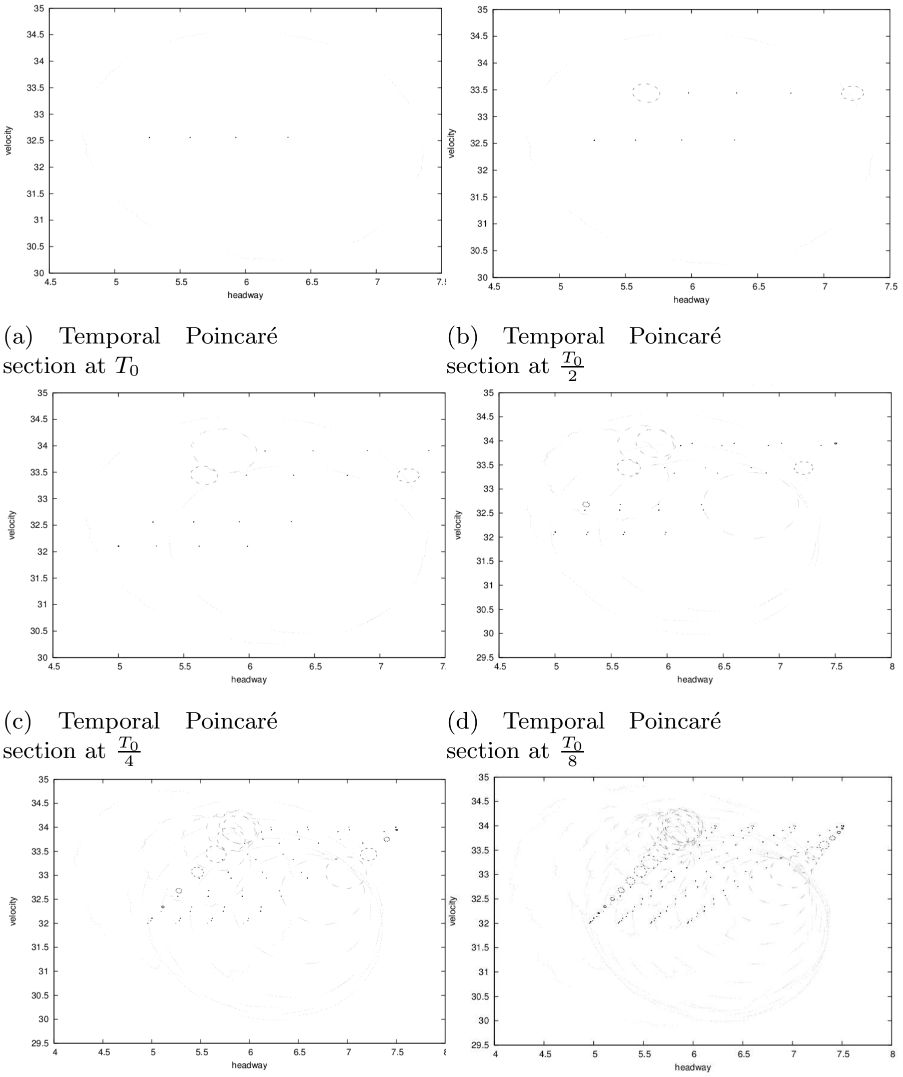


Figure 6: State space attractor for car 2



(a) Temporal Poincaré section at T_0

(b) Temporal Poincaré section at $\frac{T_0}{2}$

(c) Temporal Poincaré section at $\frac{T_0}{4}$

(d) Temporal Poincaré section at $\frac{T_0}{8}$

(e) Temporal Poincaré section at $\frac{T_0}{16}$

(f) Temporal Poincaré section at $\frac{T_0}{32}$

Figure 7: Temporal Poincaré sections of car 2 attractor

The attractor appears to contain "chaotic fuzz", but attempts to calculate the maximal Lyapunov exponent λ were inconclusive. This is shown in Figure 8, which has fits to the linear scaling region varying embedding dimension m from 4 to 8. The plots were generated using the Rosenstein algorithm[9] as implemented in TISEAN[6]. The embedding delay was chosen as the first minimum of mutual information, and the embedding dimension was initially estimated using false nearest neightbor (also both using TISEAN). Note that the values printed in

m	λ
4	0.0030
5	0.0025
6	0.0021
7	0.0018
8	0.0015

Table 3: Lyapunov estimation for car 2 attractor

the figure are the different fit slopes, e.g., λ_4 refers to the fit for λ with $m = 4$, not to the 4th Lyapunov exponent. As Table 3 shows, there is no range of m values for which the scaling region has constant slope. Attempts to calculate the correlation dimension were similarly problematic.

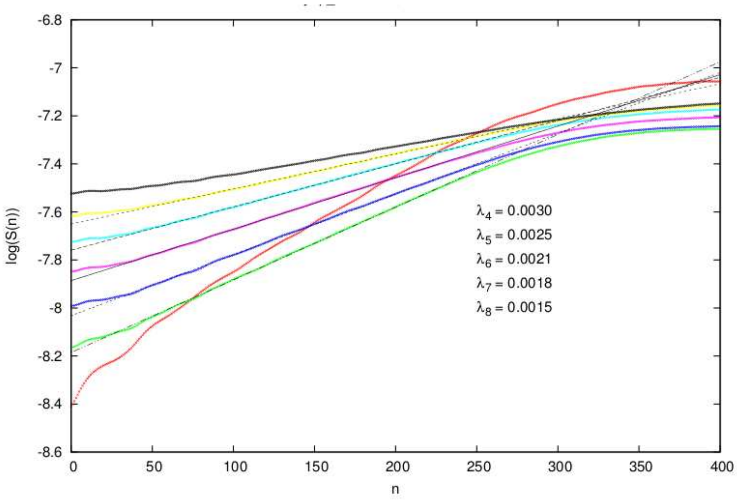


Figure 8: Lyapunov estimation for car 2 attractor

4.3 Car position

The previous sections all look at the behavior of only the second car. In fact, the dynamics begin to change farther back from the lead car. Figures 9, 10 and 11 show the velocity trajectories of cars 2, 10 and 20 respectively, from the simulation with $V_0 = V = 20ms^{-1}$. Unlike car 2, cars 10 and 20 occasionally come to complete stops. Figure 12 shows the velocity distribution for each car in this simulation, which shows that the cars farther back in the line systematically have wider velocity distributions. I attempted Lyapunov exponent calculations for cars 10 and 20 as described in Section 4.2 but the calculations were similarly unsuccessful.

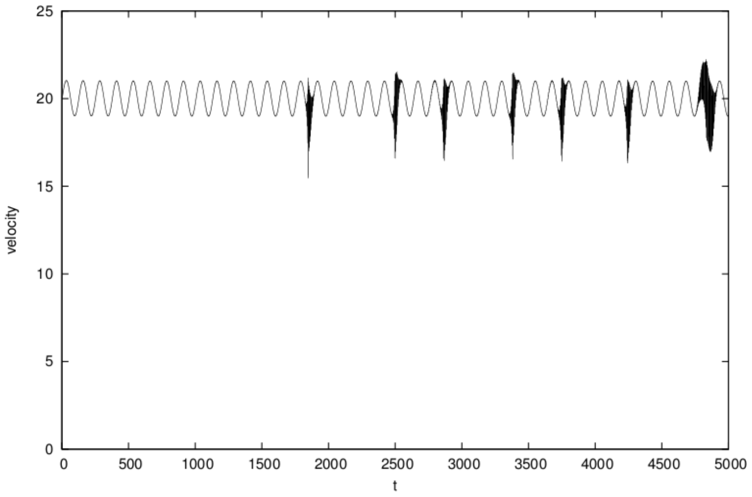


Figure 9: Velocity trajectory of car 2

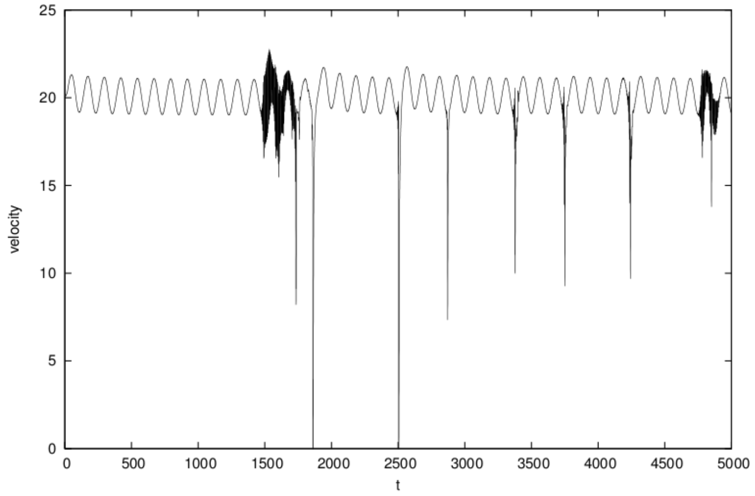


Figure 10: Velocity trajectory of car 10

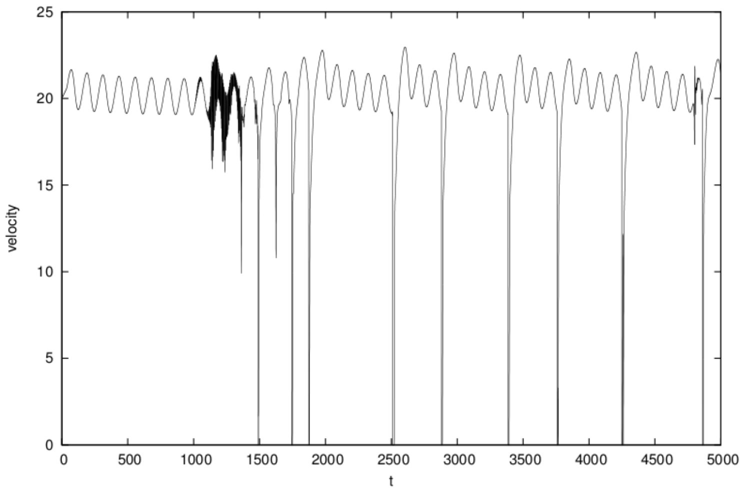


Figure 11: Velocity trajectory of car 20

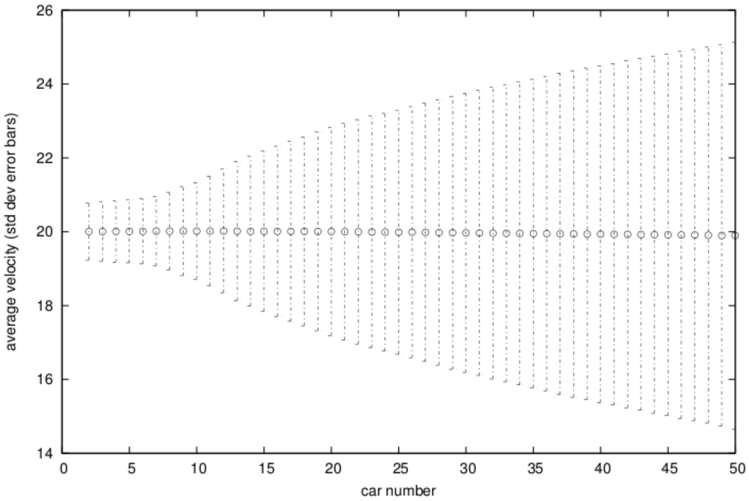


Figure 12: Velocity distributions of cars 2 through 50. Error bars show standard deviation.

5 Conclusions

It is a limitation of the design of car-following models that they cannot be used to compare traffic with a pace car to traffic without a pace car. Furthermore, this is a very approximate model of real traffic and has calibration issues with choosing the model parameters, as described in Sections 2.4 and 3.3. A more thorough exploration of the parameter space would be an interesting topic for future research. Nonetheless, to the extent that the model is valid, it does suggest one safety benefit of pace cars (other than the obvious safety benefit of lower velocities). Specifically, the cars farther back in a group of coupled cars experience more erratic behavior and have wider velocity distributions. From this perspective, pace cars decouple one group of cars from the next, which according to this car-following model, should result in more predictable trajectories. However, attempts to quantify the idea that the behavior of the back cars was "more chaotic" or "less predictable" were unsuccessful.

References

[1] Nicola Bellomo and Christian Dogbe. On the modeling of traffic and crowds: A survey of models, speculations, and perspectives. *SIAM Rev.*, 53(3):409–463, August 2011.

[2] R. L. Bierley. Investigation of an inter vehicle spacing display. *Highway Research Record*, (25):58–75, 1963.

[3] Mark Brackstone and Mike McDonald. Car-following: a historical review. *Transportation Research Part F: Traffic Psychology and Behaviour*, 2(4):181 – 196, 1999.

[4] Robert E. Chandler, Robert Herman, and Elliott W. Montroll. Traffic dynamics: Studies in car following. *Operations Research*, 6(2):pp. 165–184, 1958.

[5] Denos C. Gazis, Robert Herman, and Renfrey B. Potts. Car-following theory of steady-state traffic flow. *Operations Research*, 7(4):pp. 499–505, 1959.

[6] R. Hegger, H. Kantz, and T. Schreiber. Practical implementation of nonlinear time series methods: The tisean package. *CHAOS*, 9:pp. 413–435, 1999.

[7] Liu Lijun, Ma Hongxia, and Li Song. Transformation of chaos in traffic flow based on distance headway. In *Proceedings of the 2010 International Conference on Intelligent Computation Technology and Automation - Volume 03*, ICICTA '10, pages 1095–1098, Washington, DC, USA, 2010. IEEE Computer Society.

[8] Colorado Department of Transportation. I-70 pacing. `http://www.coloradodot.info/travel/i-70-pacing.html`, May 2012.

[9] Michael T. Rosenstein, James J. Collins, and Carlo J. De Luca. A practical method for calculating largest lyapunov exponents from small data sets. *Physica D: Nonlinear Phenomena*, 65(12):117 – 134, 1993.

[10] J.C. Sprott. A simple chaotic delay differential equation. *Physics Letters A*, 366(45):397 – 402, 2007.

Fractal Image Compression

Yevgen Matviychuk

University of Colorado at Boulder

Department of Electrical, Computer, and Energy Engineering

This work studies the ways of incorporating self-similarity, found in many commonly considered images, in solving the problem of their efficient representation. Strengths and weaknesses of general basic fractal image compression method with several modifications were analyzed. Finally, the compression algorithm based on the sparse approximation by the orthogonolized elements of the image-specific dictionary was developed.

Introduction.

All digital acquisition and display devices treat visual information as a number of points, pixels, on a grid. It is the most natural way for humans to interact with computer. However, since subjects of almost all of the images possess some sort of structure on larger scales, this representation is not efficient and often highly redundant. Therefore, the main task of image compression is to describe the important information contained in the image as briefly as possible but such that this description would be sufficient to reconstruct the image as close to the original as possible.

This work focuses on rather unconventional approach to the image compression problem, which, however, is based on very natural assumption that a lot of real world objects (and, therefore, their images) are almost self-similar, or statistically fractal. Trees, clouds and mountains are just several common examples of such objects. Obviously incorporating self-similarity property may simplify their description.

Methods discussed in this work were tested on the 256 × 256 pixels grayscale images, however obtained results can be generalized for color images of any size as well.

Mathematical background.

Most of the widely used image compression methods are based on some kind of transform (Discrete Cosine Transform, Wavelet Transform, etc.), which is equivalent to changing the representation basis. Finding a sparse description of image in some basis allows ignoring all but the highest coefficients, which effectively reduces the amount of information without significant loss of quality. This technique gives good results in most cases. Furthermore, existence of fast computational algorithms makes their implementation efficient.

In contrast, the Fractal Image Compression method provides a completely different solution to the problem. Notice that the most of natural (photographic) images often have repeated or similar regions (Figure 1.a). This feature is even more obvious in the images that contain patterns or textures (Figure 1.b-c). Obviously, exploiting this fact may lead to more efficient image representation just by describing only one such region and then distributing it to the other parts of the image.

In other words, an image often can often be considered as a fractal, an object, iteratively constructed from the transformed parts of itself that has very complicated structure but relatively simple mathematical description. Therefore, compression can be achieved by encoding the rule that is used to generate the fractal. Notice, that similarity only with the parts not the whole image is assumed. This is one of the differences of such representation from true geometrical fractals. Another is the fact that self-similarity in images is usually not exact. This means that the fractal description of image will inevitably introduce some amount of error or lead to information loss in case of compression.

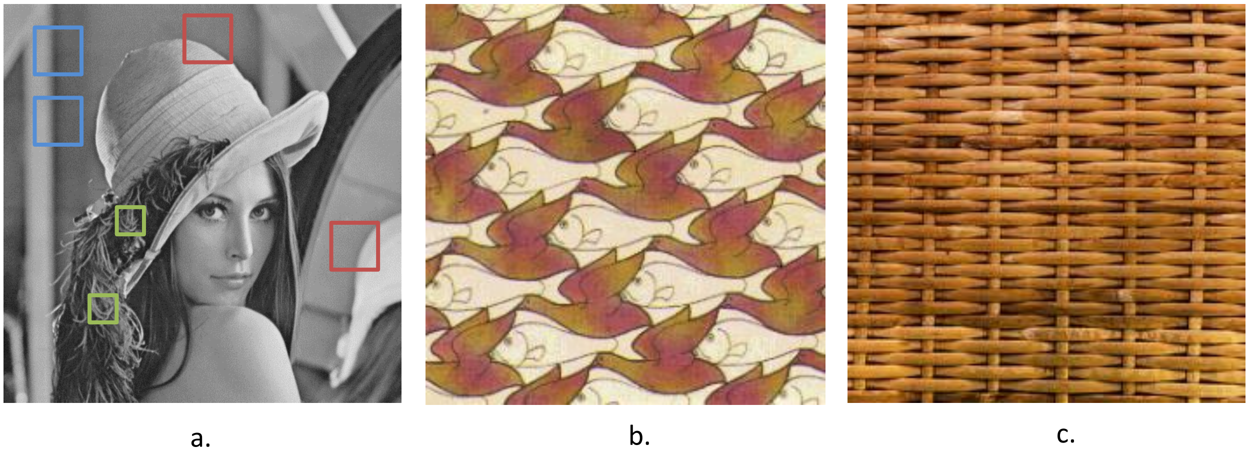


Figure 1. – Self-similarity in natural (photographic) images (a), patterns (b) and textures (c).

The encoding procedure, which is essentially the derivation of the fractal description, is formalized by the following algorithm [8, 9].

1. Partition the whole image I into disjoint set of N range regions R_i, $i = 1 \ldots N$.

2. In the same image I define M possibly overlapping domain regions D_j, $j = 1 \ldots M$.
3. Define K transformations on the set of domain regions T_k, $k = 1 \ldots K$.
4. For each range region R_i find the pair consisting of the domain region D_j and transformation T_k such that the Mean Squared Error weighted by the number of pixels and defined as:

   $$MSE = \left\| c \cdot T_k(D_j) + b - R_i \right\|_2 / n^2 \tag{1}$$

   is minimized. Moreover, additional parameters b and c that correspond to the offset and scaling are used for brightness and contrast adjustment respectively [6]. By solving the least squares minimization problem defined by (1), they can be expressed as:

   $$c = \frac{n^2 \cdot T_k(D_j)^T \cdot R_i - (T_k(D_j)^T \cdot 1_{n^2}) \cdot (R_i^T \cdot 1_{n^2})}{n^2 \cdot T_k(D_j)^T T_k(D_j) - (T_k(D_j)^T \cdot 1_{n^2})^2}, \tag{2}$$

   $$b = \frac{1}{n^2}\left(R_i^T \cdot 1_{n^2} - c \cdot T_k(D_j)^T \cdot 1_{n^2}\right), \tag{3}$$

   where n^2 is a number of pixels, 1_n^2 is an $n^2 \times 1$ vector of ones $[1, 1, \ldots, 1]^T$.

The result of this algorithm is the collection of transforms $W : F \rightarrow F$, where $F = \{I : Z^{n \times n} \rightarrow \Re\}$ denotes the space of n×n images:

$$W(\cdot) = \cup_{i=1}^{N} w_i(\cdot), \tag{4}$$

$$w_i(\cdot) = R_i = T_k(D_j), \;\; i = 1 \ldots N, j = 1 \ldots M, k = 1 \ldots K. \tag{5}$$

Equations (4) and (5) assume that each range and domain region can also be represented as a vector in $\Re^{n \times n}$ by masking corresponding part of the image and setting all the other pixels to 0.

The obtained collection of transforms (4) is called an Iterated Function System, IFS (or, more precisely, Partitioned Iterated Function System, PIFS, since transforms are performed on the parts of the original image). It describes the image I as a union of its transformed parts.

The fundamental property, which the PIFS W should satisfy in order to converge to a fractal object, is contractivity. The transformation $W : X \rightarrow X$ on the metric space (X, d) is said to be contractive if there exists a constant $0 \leq C < 1$, called the contractivity factor, such that:

$$d(W(x), W(y)) \leq C \cdot d(x, y), \tag{6}$$

where $x, y \in X$, $d(\cdot, \cdot)$ is a metric on X.

Note, that the space of images F is complete with respect to the supremum metric, which has meaning of the maximum pixel-wise difference between two images, and is defined as [1]:

$$d_{sup}(I,J) = sup_{x,y \in Z^{\mathrm{n\times n}}} |I(x,y) - J(x,y)|. \tag{7}$$

Completeness of the space *(F, d_{sup})* allows applying the Contractive Mapping Theorem to the PIFS *W*, which means that there is a unique fixed point, I^* in the space of images *F*, such that $W(I^*) = I^*$. This fixed point is often called the attractor of PIFS, in terms of the fractal compression theory. The important property of this contractive mapping is that starting with any point $I_0 \in F$ and running the iterative procedure: $I_1 = W(I_0)$, $I_2 = W(I_1) = W(W(I_0))$, $I_n = W(\ldots W(I_0))$, I_n will eventually converge to the fixed point of W (usually, in practice, in no more than 7-10 iterations):

$$I^* = \lim_{n\to\infty} W^n\,(I_0). \tag{8}$$

The contractivity property of PIFS gives the background for the decoding algorithm. The encoded image can be reconstructed (up to some tolerance ε) by iteratively computing the attractor of PIFS I^*. Therefore, the problem of fractal image compression can be formulated as finding the PIFS, whose attractor would be as close to the original image as possible.

The reconstruction error is commonly measured by the Man Squared Error, defined as the Euclidean norm of the difference between the original *I* and reconstructed $\hat{I}$ images:

$$MSE = \sqrt{\sum_{i,j}\left(I_{ij} - \hat{I}_{ij}\right)^2}, \tag{9}$$

or in logarithmic units as the peak signal-to-noise ratio:

$$PSNR = 20 \cdot \log_{10}\left(\frac{\max_{ij} I_{ij}}{MSE}\right). \tag{10}$$

Moreover, the coding rate R is often used to assess the efficiency of compression. It can be defined as an average number of bits needed to describe each pixel by using particular coding scheme. Uncompressed 256-levels grayscale image has a coding rate of 8 bits per pixel (bpp). The coding rate achieved by the fractal image compression method can be estimated as follows:

$$R = \frac{(2\cdot 7 + \log_2 M + 3 + 2\cdot 8)\cdot N}{256^2}. \tag{11}$$

Equation (11) assumes that to store the fractal code for a 256 × 256 - pixels image up to 7 bits would be needed to encode both, horizontal and vertical position for each of N range regions. Furthermore, $\log_2$M bits are required to store the identifier of chosen domain region, 3 bits (if using only affine transformations, or $\log_2$K, in general) for the code of applied transformation, and finally 8 bits to encode brightness and contrast parameters.

Notice that these values are approximate and may not be optimal. Obviously, the number of bits used to encode the locations of the range and domain regions would depend on their number and partitioning scheme. Furthermore, PIFS information can be stored more compactly using special coding methods. For example, since distributions of the contrast and brightness are likely to be non-uniform, using entropy encoding may reduce the number of bits, needed to store these values, down to 7 and 5 bits respectively [1]. However, these techniques were not considered in this project and the coding rate parameter defined by (11) is used only to compare the performance of different modifications of the basic algorithm.

Common methods.

Although running the described basic algorithm is sufficient to construct the PIFS, which will eventually converge to the original image, it does not give any specific directions on partitioning the image or defining transforms. Choosing these parameters plays a crucial role in the overall performance of image compression procedure. Moreover, since the search algorithm is naturally very computationally expensive, optimizing its running time is another problem to consider.

The first and most obvious way of improving the reconstructed image quality is to reduce the size of the range regions. This approach, indeed, gives very good results as shown on Figure 3. However, besides of rapid MSE decrease, smaller size of the range regions requires storing more information about PIFS transforms. This inevitably increases coding rate (see Figure 4). In fact, for the range regions of size of 2 × 2 pixels the coding rate has exceeded 10 bpp, which makes the compression completely meaningless (as compared to the original 8 bpp rate).

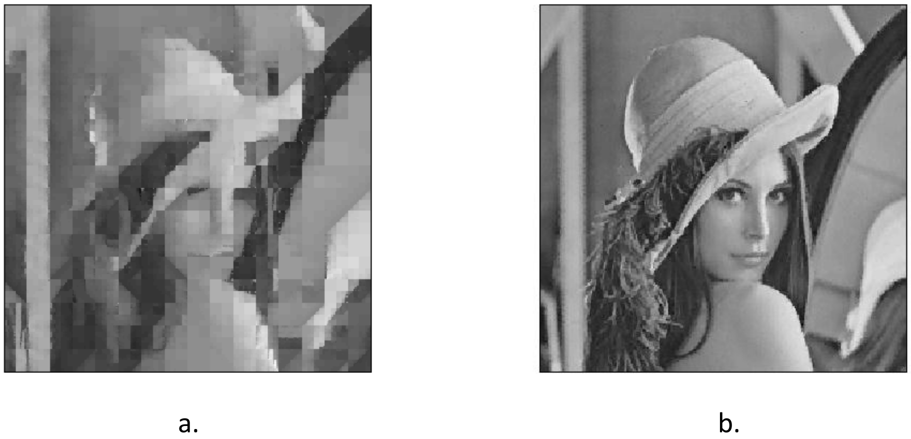


Figure 3. – Reconstruction results using 16 × 16 (a.) and 4 × 4 pixels (b.) range regions.

To avoid this effect, image-dependent segmentation (as opposed to image-independent tiling) can be used [9]. Even though, the second method is simpler to implement, it does not distinguish the inherent structure of the image – one of the fundamental assumptions of the fractal compression. Using larger range regions and reducing their size only where it is needed (for the parts of the image with fine detail, for example) allows significantly decreasing the coding rate while maintains the quality on acceptable level. Most of the image-dependent partitioning algorithms are based on the usual tiling strategy but with dynamical splitting criterion, and differ mostly in the shape of produced partitions.

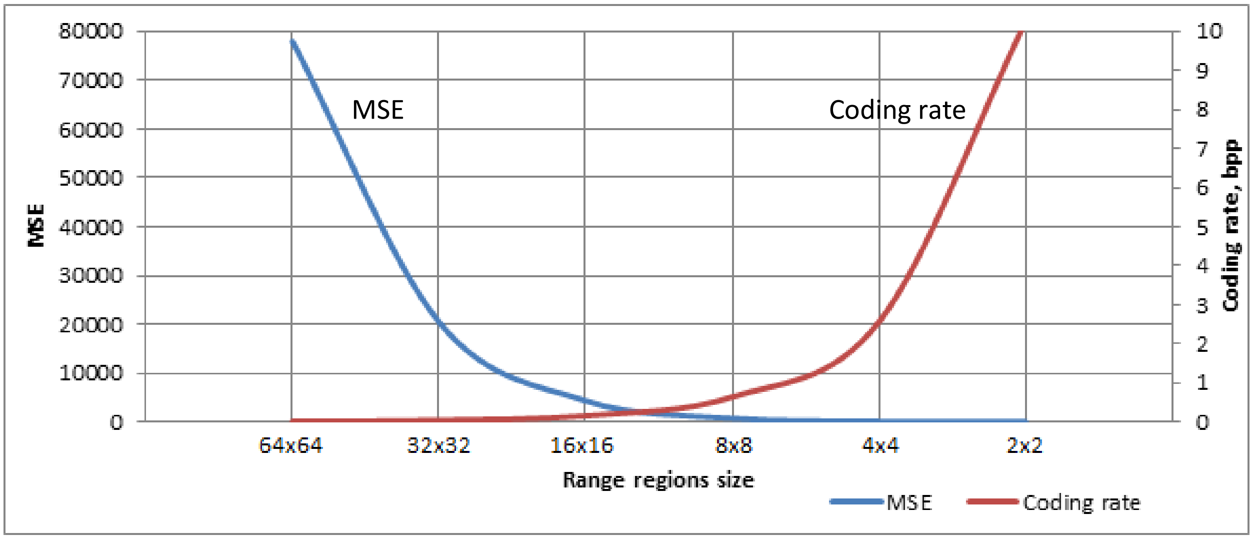


Figure 4. – Reconstruction MSE and coding rate as functions of the range region size.

The simplest most commonly used partitioning algorithm, the Quadtree (QT) Partition, is based on the square-shaped tiles. If considered range region can not be approximated by any element of domain pool such that the matching error is below some predefined threshold ε_{QT}, this result is discarded and range region is split into 4 new square regions of smaller size; the matching procedure is then repeated for each of them separately. Notice that this does not require adding new domain regions, but only adjusting the downsampling parameters for existing ones.

This algorithm was used in this project; its result (using 2 × 2 minimum partition size) is shown on the Figure 5. Notice that larger tiles were sufficient to represent the wall on the background or the shoulder, which are the uniformly filled parts of the picture. However, the feathers on the hat required much finer partitioning. Remarkably, even though the MSE increased (39.3 as opposed to 0.55 in the uniform partitioning case) reconstruction quality is still very high. In addition, the coding rate was reduced more than twice (from 10.25 to 3.93 bpp).

It's worth noting that the QT Partitioning scheme is far from optimal. Details of images are not restricted to occupy square regions but instead are spread in different shapes and directions. The inefficiency of the partitioning is visible along the sharp edges in the image (Figure 5), which tend to be represented by unnecessarily large number of small partitions. There exist several modifications of the QT algorithm that account for this effect – for example, splitting the square region only in one direction, which effectively creates two horizontal or vertical rectangles leads to the HV-partition. This method might have solved the problem of approximating sharp edges of the image. To account for the details in directions other than horizontal or vertical, the triangular partition can be used. Moreover, Evolutionary Partitioning, which is the more elaborate method based on the idea of not only splitting but also merging similar neighboring tiles often gives the most natural partitioning that depends on the image content. The obvious problem with all these methods is their complexity and the difficulty of implementing an efficient searching algorithm in the increasing pool of domain regions.

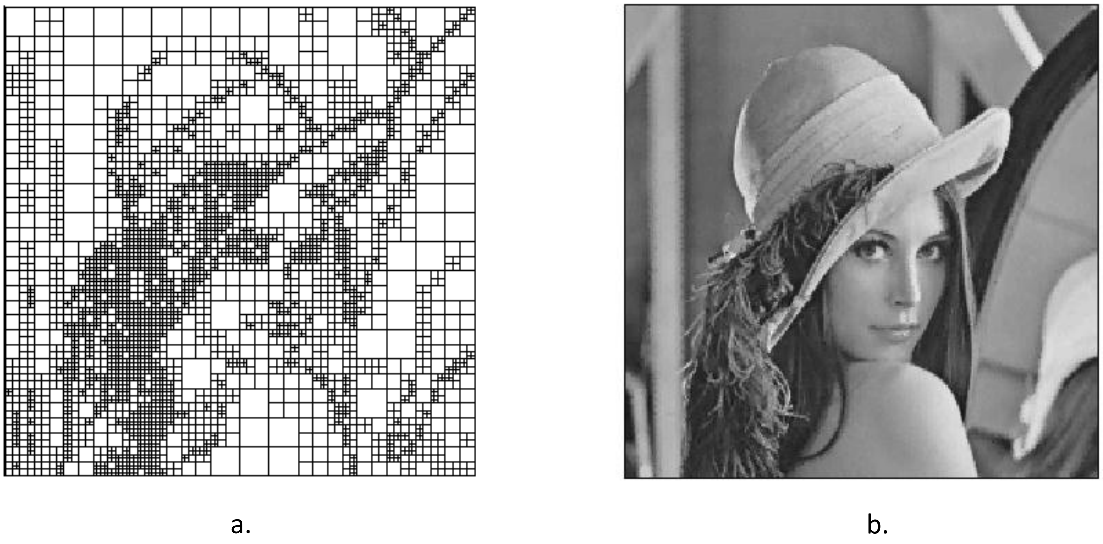


Figure 5. – QT Partition of the image with minimum tile size of 2 × 2 pixels: partition boundaries (a) and reconstructed image (b).

In fact, finding the match between each range region and the transformed domains is the most computationally expensive part of the compression algorithm. As a rule of thumb, the size of the domain regions is usually chosen to be greater than the size of the range region. This requires downsampling in addition to the spatial transformation, which ensures the contractivity property of the transform. However, this is not a strong requirement [1]. Then considering the n × n pixels image and m × m range region, the maximum number of possible domains can be estimated as:

$$M_{max} = \sum_{i=m+1}^{n-1}(n-i+1)^2 = \frac{1}{6}(n-m-1)(2m^2-4mn-5m+2n^2+5n+6). \quad (12)$$

Fortunately, there is no need to search through all of these domains, because some of them are likely to be very similar. This assumption underlies the efficient searching algorithms, which first classify the members of the domain pool (for example, into those with smooth gradients or sharp edges [2,4]) and then perform the search in the appropriate class according to the features of the range region to be matched.

The transforms T_k are usually chosen to be linear, which includes rotations in multiples of 90 degrees, and reflections with respect to different axes. Although they give the best results for the wide variety of images, it should be noticed that there are also non-linear transforms in the form of $R = D^p, 0 < p < 1$ proposed for use, for example, in the fractal compression of sound signals [4]. These may be suitable for compression of certain types of images as well.

Used approach and its results.

Most of the existing fractal image compression algorithms and of all the previously mentioned modifications to the basic algorithm assume that only one transformed domain region is chosen to match the particular range region. In contrast, the main conjecture in this work is that better compression performance can be achieved by using a linear combination of several domain regions. Similar ideas can be found in [1, 5].

To implement this approach, given the domain pool $\{D_j, j = 1 \ldots M\}$ and the set of corresponding transforms $\{T_k, k = 1 \ldots K\}$, let us first define the dictionary V as the set, which consists of results of applying all the transforms T_k to each of the domains D_j: $V = \{D\} \times \{T\}$. If each transformed domain region $T_k(D_j)$ is viewed as a vector in n × 1 dimensional space, the dictionary V can be represented as an $n \times K \cdot M$ matrix. Then the approximation to the considered range region R_i is given by:

$$\widehat{R_\iota} = V \cdot \alpha + b, \quad (13)$$

where α is a $K{\cdot}M \times 1$ vector of representation coefficients, and b is a brightness adjustment parameter. Therefore, the problem is reduced to finding the coefficients α such that the weighted MSE $\|R_i - \widetilde{R_\iota}\|_2/n^2$ is minimized. Moreover, in order to obtain efficient compression, vector α should have all but a few entries equal to zero (α should be sparse).

In general, since dictionary V defines a non-orthogonal basis, finding the optimal sparse representation of R_i in terms of its elements is an NP-hard problem. However, greedy algorithms, such as Matching Pursuit, can find suboptimal sparse solution. Its modification, the Orthogonal Matching Pursuit (OMP, also known as the Gramm-Schmidt Orthogonalization) converges in a finite number of iterations, and, therefore, gives a more accurate solution faster. The idea of the OMP algorithm is to form an orthonormal basis from those elements of overcomplete non-orthogonal dictionary that capture the most of the energy of the given approximated vector.

The following realization of OMP was implemented:

1. Given the range region vector R_i and non-orthogonal dictionary V, define the remainder to be: r = R_i.
2. Find the element in V (v_0, $n \times 1$ column vector) that approximates the remainder r with the smallest MSE by evaluating the $argmax_i|\langle r\ ;\ v_i\rangle|$, the inner product between those vectors. Computation-wise this operation is performed by multiplying the vector r with the matrix V.
3. Define the orthogonal dictionary as $U = \{v_0\}$.
4. Update the remainder: $r = r - \langle r\ ;\ v_0\rangle \cdot v_0$.
5. Find the vector v_1, similarly to v_0, as described by the step 2.
6. Find the new vector u_1, orthogonal to each vector in U: $u_1 = v_1 - U \cdot U^T \cdot v_1$; and normalize it: $u_1 = \frac{u_1}{\|u_1\|}$. Update the matrix U by adding this new vector to it.
7. Update the remainder: $r = r - \langle r\ ;\ u_1\rangle \cdot u_1$.
8. Repeat from step 5 until the norm of the remainder $\|r\|$ is less than some predefined threshold ε_{max}, or the number of elements in the dictionary U exceeds N_α.

Given the orthogonal dictionary (basis) U, the representation coefficients of the vector R_i can be obtained by calculating the inner products with the elements of U:

$$\beta = R_i^T \cdot U. \tag{14}$$

Then the coefficients α in the original basis V can be found by the iterative procedure starting from the last taken vector (here $\hat{v}_i$ denotes the vector in V taken on the i-th step of the algorithm):

$$\alpha_i = \frac{\beta_i}{\langle \widehat{v_i} \, ; u_i \rangle}; \tag{15}$$

$$\beta_{1\ldots i-1} = \beta_{1\ldots i-1} - \alpha_i \cdot \langle \widehat{v_i} \, ; \, u_{1\ldots i-1} \rangle, \text{ for } i = N_U \ldots 1. \tag{16}$$

Notice that this algorithm has two parameters that eventually affect sparsity of α: maximum approximation error ε_{max} and maximum number of non-zero coefficients N_α. In other words, the algorithm will add new approximation coefficients until either their number exceeds N_α or acceptable accuracy of approximation is achieved, $\left\|R_i - \widetilde{R_\iota}\right\|_2 \leq \varepsilon_{max}$. Varying these parameters along with the Quadtree Partitioning threshold ε_{QT} gives more possibilities to optimize the compression performance in terms of the MSE and coding rate. Figure 6 represents the fragment of the algorithm flowchart that explains the roles of these parameters.

The following table compares compression results obtained by three different methods: uniform range regions partitioning with the smallest partition equal 2 × 2 pixels, the QT partitioning with the same smallest level, and the QT partitioning with the 4 × 4 pixels smallest partition and the sparse approximation in the domain pool. Notice that the sparse approximation approach has improved both the MSE of reconstruction and the coding rate. Moreover, the computation speed has increased due to the coarser partitioning and using more efficient searching algorithm (computing the inner products via matrix multiplications instead of pairwise comparison).

Table 1. – Comparison of different coding algorithms.

Method	MSE	Coding rate
Uniform 2x2 partition	0.55	10.25 bpp
Using QT only (min level – 2x2 pixels)	39.3	3.93 bpp
Sparse Approximation and QT (min level is 4x4 pixels)	26.5	3.54 bpp

In general, more basis vectors are needed to approximate larger range regions. The reason for this, besides of the increased dimensionality of the vector space, is that there are fewer ways to define the pool of distinct domain regions of larger size. Therefore, it may not be optimal to limit the number of approximating coefficients by the same constant N_α on different levels of the QT partition.

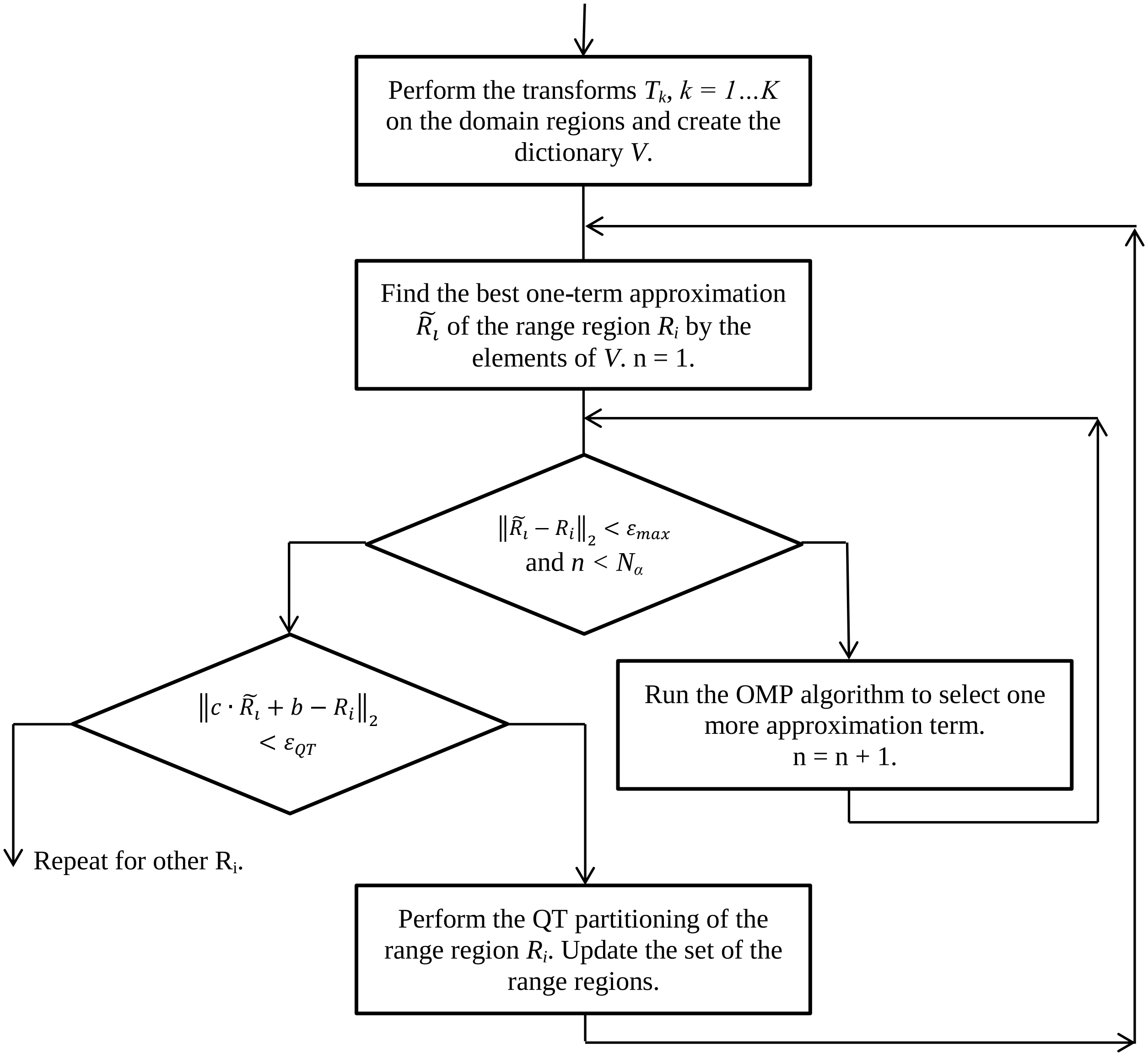

Perform the transforms T_k, $k = 1 \ldots K$ on the domain regions and create the dictionary V.
Find the best one-term approximation $\widetilde{R_i}$ of the range region R_i by the elements of V. n = 1.
$\|\widetilde{R_i} - R_i\|_2 < \varepsilon_{max}$ and $n < N_\alpha$
Run the OMP algorithm to select one more approximation term. n = n + 1.
$\|c \cdot \widetilde{R_i} + b - R_i\|_2 < \varepsilon_{QT}$
Repeat for other R_i.
Perform the QT partitioning of the range region R_i. Update the set of the range regions.

Figure 6. – Fragment of the compression algorithm flowchart, corresponding to the fitting of the range region Ri by the elements of the dictionary V. It is assumed that the range and domain pools have been defined already.

As an illustration for this hypothesis, consider the plot of the MSE versus the number of domain pool elements of different scales (Figure 7). It can be noticed that there are several ways to obtain the same reconstruction quality. For example, uniform 4 × 4 partitioning with each range region approximated by only one domain gives the same MSE ≈ 67, as the uniform 8 × 8 partitioning with 6 approximation coefficients, or 16 × 16 partitioning with 65 approximation coefficients. However, these three schemes differ in the resulting coding rate. Using finer partitioning, as was previously shown, increases the number of parameters to be stored (positions of each region). On the other hand, large scale regions can be defined by fewer parameters of shorter lengths and, therefore, have more compact representation.

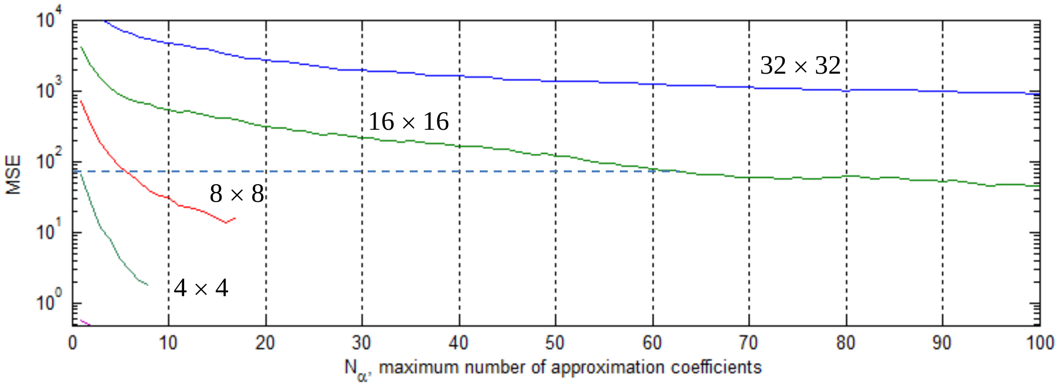


Figure 7. – MSE versus the number of approximation coefficients for different sizes of R_i.

Compression results obtained by using the range regions of different sizes (without QT partitioning) are summarized in Table 2. The maximum number of coefficients is chosen such that the coding rates are approximately equal. The MSE threshold for sparse approximation was set to ε_{max} = 0.5 to encourage the algorithm to take all N_α representation coefficients.

Table 2. – Compression results using the uniform partitioning and sparse approximation.

Size of the range regions	Number of approximation coefficients	MSE	Coding rate
32 × 32	145	866.7	2.31 bpp
16 × 16	38	179.35	2.45 bpp
8 × 8	8	39.19	2.32 bpp
4 × 4	1	66.86	2.45 bpp
2 × 2	1	0.55	10.25 bpp

Based on these observations, the maximum number of approximation coefficients (parameter N_α) was made dependent on the region dimensions, as defined in the Table 3. Now the algorithm first tries to approximate the larger range regions and performs quad tree partitioning only if there exist as good representation by only one coefficient on the finer scale. The compression results obtained by this modified algorithm are: MSE = 36.2, coding rate = 2.15 bpp, i.e. both characteristics were improved.

Table 3. – The maximum number of approximation coefficients (N_α) for regions of different sizes.

Size of the range region, pixels	2 × 2	4 × 4	8 × 8	16 × 16	32 × 32	64 × 64	128 × 128
N_α	3	3	8	50	150*	300*	600*

* - in practice these values are constrained by the number of dictionary elements.

Finally, let us compare performance of the designed fractal image compression algorithm with the conventional transform coding. Consider the two types of linear transformations: Discrete Cosine Transform, used in the JPEG compression algorithm and Wavelet Transform (with Haar and Daubechies bases). The number of representation coefficients produced by each of them is equal to the number of pixels (i.e. $256^2 = 65536$, in this case). Therefore, 16 bits are needed to store each coefficient’s location and 8 bits to store its value (assuming that 256-level quantization scheme is used). In order to achieve the same coding rate as for the fractal compression (2.15 bpp in the best case), only the highest $N_c = \frac{2.15 \cdot 65536}{24} \approx 5900$ coefficients should be stored. Table 4 represents the reconstruction MSE values for these cases.

Table 4. –MSE achieved by different compression methods with the same coding rate.

Type of the transform	Achieved MSE
Discrete cosine transform	51.75
Wavelet transform (Haar wavelets)	38.08
Wavelet transform (Daubechies-8 wavelets)	25.56
Fractal compression	36.2

These data show that performance of the fractal compression method is comparable to the most popular transform-based compression approaches. Moreover, implemented algorithm can be further improved by more sophisticated partitioning and searching schemes. However, from the other side, increased complexity of the fractal encoding algorithm is one of the main reasons that prevent it from being widely used.

Conclusion.

The main result of this work is design and implementation of an image compression algorithm based on the self-similarity properties of images. The basic fractal image compression algorithm is defined such that it allows ample number of improvements and modifications. Several possible approaches to optimize the partitioning scheme and searching algorithm were analyzed. The final implemented version of the algorithm uses Quad Tree partitioning to define the range regions. This method effectively determines highly detailed parts of the image that may need more accurate approximation. To make searching and fitting procedures more efficient, several domain regions are used to approximate each range region. They are drawn from the domain pool by performing the Gramm-Schmidt orthogonalization. This approach has given results comparable to the usual compression methods, which suggests that possibly using hybrid (for example, wavelet-fractal, [7]) methods would give further improvement of reconstruction quality.

Although, because of the complexity and high computational cost of the encoding algorithm, fractal image compression methods are unlikely to find a direct implementation in compression of the images of photographic quality, they are interesting as a concept itself. Moreover, image representation that they produce has several unique useful features. One of them is the resolution independence, also known as fractal zoom, which allows increasing image size during the reconstruction. Of course, no new information will be added, but, instead of simply deducing the unknown pixels by averaging, fractal based reconstruction would try to preserve the same complexity on the finer scales. Therefore, it may be beneficial to use such methods to represent the images (or even only their parts) that contain some sorts of patterns or textures. Particularly, they may be useful in situations, where the overall pattern structure is more important than the particular values of each pixel (for example, patterns formed by sand grains, or trees in the forest viewed from the distance). Obviously, another quality metric based on the subjective perception instead of the MSE would be needed to estimate the performance in this case.

Bibliography

1. Y. Fisher, Editor, “Fractal Image Compression: Theory and Applications”, Springer-Verlag, 1994.
2. A.E. Jacquin, “Image coding based on a fractal theory of iterated contractive image transformations”. IEEE Trans. on Image Processing, vol. 1, pp.18-30, 1992.
3. Dinesh Rao B., Ganesh Kamath, Arpitha K. J., “Difference based Non-linear Fractal Image Compression”. International Journal of Computer Applications (0975 – 8887). Volume 38– No.1, January 2012
4. Dan C. Popescu, Alex Dimca, and Hong Yan, A Nonlinear Model for Fractal Image Coding: IEEE Transaction on image processing, Vol. 6, No. 3, March 1997.
5. M.H. Hayes, G. Vines, “IFS Image Coding Using an Orthonormal Basis”. ISCAS, pp. 621 – 624, 1994.
6. Dietmar Saupe, Raouf Hamzaoui, Hannes Hartenstein, “Fractal Image Compression. An Introductory Overview”. Report // Universität Freiburg, Institut für Informatik. Freiburg : Univ., Inst. für Informatik, 1997.
7. Ramyachitra Duraisamy, L. Valarmathi and Jeyalakshmi Ayyappan, “Iteration Free Hybrid Fractal Wavelet Image Coder”, International Journal Of Computational Cognition, Vol. 6, No. 4., December 2008.
8. M. F. Barnsley and L. P. Hurd, Fractal image compression, A. K. Peters, Boston, 1992.
9. Ning Lu, Fractal Imaging, Academic Press, 1997.
10. Jin Li, C.-C. Jay Kuo, Hybrid wavelet-fractal image compression based on a rate-distortion criterion. SPIE Proceedings: Visual Communications and Image Processing, 1997.
11. A.Muruganandham, S.Karthick, Dr. R.S.D. Wahida Banu, A Fast Fractal-Curvelet Image Coder. International Journal of Computer Applications (0975 – 8887) Volume 35– No.12, December 2011.
12. Soyjaudah, K. M. S.; Jahmeerbacus, I., “Fractal image compression using quadtree partitioning”. International Journal of Electrical Engineering Education; Jan2002, Vol. 39 Issue 1, p71.

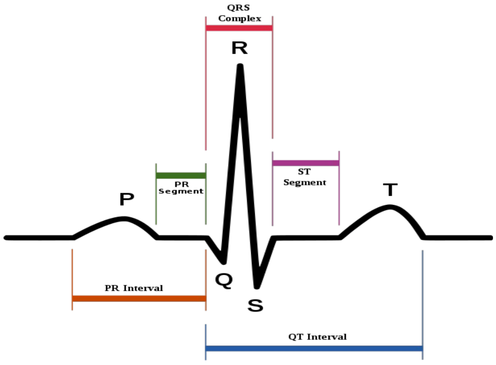
Adam Rose
Chaotic Dynamics Final Paper

Chaotic Dynamics Indicators Applied to EKG Data

Abstract: Cardiac arrhythmia are characterized by irregularities in the electrical activity of the heart. Many arrhythmia are deadly, and most may be diagnosed from the output of an electrocardiogram. In this paper, EKG data from the MIT-BIH database are analyzed with a variety of tools in an effort to ascertain the objective dynamical properties of harmful arrhythmia. In particular, we employ tools from the TISEAN 3.0.1 nonlinear time series analysis toolkit, the Grassberger Procaccia algorithm, a heuristic method for identifying chaos from frequency content, and observations from a mean squared error histogram method. The robustness and applicability of these methods are compared.

Introduction: Electrocardiogram (EKG or ECG) is powerful diagnostic tool. By simply attaching electrode to the surface of the skin and recording the output on a device external to the patient, one can obtain a reasonable assessment the heart rate and regularity of heart beats.

Figure 1



As of 2002 (Owis et. al.), the conventional method of detecting cardiac arrhythmia was by human observational of an EKG. By comparing observed output to the well documented electrical activity of a healthy heartbeat, one is able to diagnose potential problems. However, the inadequacy of such a method is clear. With the plethora of patients with heart disease and the volume of data, it is frequent that such irregularities are not caught in a timely fashion. While not all arrhythmia are life threatening (or even harmful), as of 2011 (Gothwal et. al.) cardiac arrhythmia essentially accounted for 90% of cases of "sudden death" in which a death occurs within an hour of first symptoms. 80% of these cases were due to the arrhythmia ventricular fibrillation, often brought on by ventricular tachycardia. For these reasons, much research was done to develop automated EKG interpreters. Most of these methods involve feature classification, in which the EKG data is denoised and the component structures of regular wave (see figure1) identified and analyzed. In this paper, we seek to use alternative diagnostic tools; we apply indicators used to test for chaos in dynamical systems to detect arrhythmia.

Chaos is defined as the output of a deterministic dynamical system that exhibits sensitivity to initial conditions. As the true equations that dictate the motion of the heart are unknown to us, the question of whether the typical motion of the heart is "chaotic" is still under debate. However, it has been shown in previous work (Casaleggio and Braiotta) that the most regular of heartbeats (sinus rhythm) tend to evoke more emphatic indicators of chaos than a case of ventricular tachycardia. While this may appear strange, we find it no less effective in deducing the difference between such cases. To the casual observer a chaotic attractor are characterized by neighboring trajectories that diverge over a short timescale while being bound within attractors, having patterns of data that tend to repeat at irregular intervals, and having highly aperiodic motion. These observations lead us to useful, though sometimes heuristic, metrics of the system.

The first calculation we would like to perform is an approximation of the Lyapunov exponent of the system. Given a point in a state space x_0, and a very nearby point $x_0+\delta_0$, we define the separation between the two points as δ_n. The Lyapunov exponent is the constant λ such that $|\delta_n| \approx |\delta_0|e^{n\lambda}$. A positive Lyapunov exponent is an indicator of a chaotic system. We would like to estimate such a parameter from the EKG data, so our first step is to attempt to reconstruct the state space.

Takens' theorem states that there exists a constant "m" greater than two times the dimension of the true state space such that a system constructed from a single variable and "m-1" uniformly delayed copies of the same variable is diffeomorphic to the true state space. This is remarkable, and highly useful, as it allows us to attempt to construct a state space diffeomorphic to the true system describing cardiac dynamics solely from EKG data.

Methodology: All data analyzed is found in the MIT-BIH arrhythmia database.

We utilize the Tisean toolkit to determine the optimal delay choice, "τ", and the embedding dimension, "m" using the functions "mutual" and "false_nearest" respectively. "Mutual" computes the Shannon entropy for a variety of box sizes. We select the box size where the first minimum of the entropy is achieved as our "τ". "False_nearest" gives useful information about the state space from the method of false-nearest-neighbors. This method is used to simulate the results of many experiments from a single trial in an autonomous system. In this method, trajectories that return within an ε-ball of a previously visited point in state space may be viewed as an entirely separate trial. Such a returning point is dubbed as a "false-nearest-neighbor," and the program "false_nearest" returns the fraction of false-nearest-neighbors present in the data for a given neighborhood size ε. We choose the embedding dimension “m” such that the fraction of false-nearest-neighbors is just under ten percent. If we call the EKG data “x(t)”, at this point we view the new state space to be constructed from the following “m” variables: x(t), x(t+ τ),...x(t+(m-1) τ).

While these methods have empirical support, there is no guarantee that the state space constructed in this way is truly diffeomorphic to the true state space—our method for calculating "m" may have failed us. However, if we have succeeded, then certain quantities including the Lyapunov exponent and correlation dimension will be the same in both the original and reconstructed state spaces. This gives us incentive to calculate these quantities for a variety of proposed values for "m," as increasing "m" slightly should not change our results. Finally, we apply the algorithms "lyap_r" and "lyap_k" as designed by Rosenstein et. al. and Kantz respectively to approximate the maximum Lyapunov exponents. Both methods take false-nearest-neighbors within a small neighborhood and estimate a spreading factor as the neighbors step forward in time. However, the way in which this small neighborhood is determined is different in each algorithm. We compare results from both algorithms.

Another parameter of interest is the correlation dimension, which we will reference as “d.” This parameter is very close to the true, box-counting dimension the attractor occupies in state space. The Grassberger-Procaccia algorithm finds an unbiased estimator

$$\hat{C}(r) = \frac{2}{N(N-1)} \sum_{i<j} \theta(r - |\mathbf{x}_i - \mathbf{x}_j|),$$

that approximates

$$C(r) = \int d\mu(\mathbf{x}) \int d\mu(\mathbf{y}) \theta(r - |\mathbf{x} - \mathbf{y}|).$$

, where "μ" is the distribution of points and "θ" is the heavy-side step function. $C(r) \sim r^d$ as $r\rightarrow 0$, so we can obtain "d" by linear regression on a plot of "log(C(r))."

We now use the work of Eckmann and Ruelle to verify that we meet some basic lower bounds on the number of points required to obtain reasonable estimations of Lyapunov exponents and fractal dimension. In particular, we can determine the maximum possible dimension attained by the Grassberger-Procaccia algorithm using “N” points of data. Given that “D” is the diameter of the state space and “r” as above, “dmax=2log(N)/log(D/r).” If we approach “dmax” from a simulation with “N” points, our results are to be disbelieved. In addition, we require at least a few false-nearest-neighbors to effectively calculate the Lyapunov exponent. Fundamentally, this means we need “N>>(D/ε)^d.” If this bound is not met by a decent margin, we must discount our results here as well.

We now turn to more heuristic methods. McDonough et. al. suggest that one could see indicators of chaos in their mean squared error histogram (Mesah) method. Their results with one and two dimensional maps show promising results. This simple algorithm appears to be resistant to noise and useful for analyzing more basic attractors with as few as two hundred points of data. The algorithm is quite simple, as stated in Table 1:

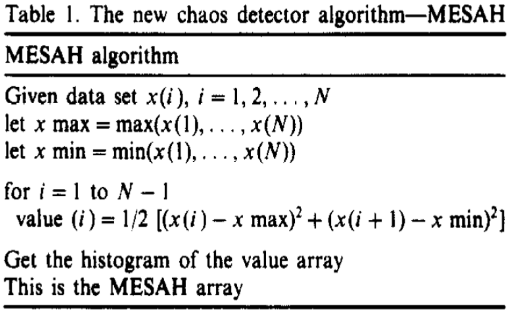


Table 1. The new chaos detector algorithm—MESAH

MESAH algorithm

```
Given data set $x(i)$, $i = 1, 2, \ldots, N$
let $x$ max $= \max(x(1), \ldots, x(N))$
let $x$ min $= \min(x(1), \ldots, x(N))$

for $i = 1$ to $N - 1$
  value $(i) = 1/2\ [(x(i) - x\ \max)^2 + (x(i+1) - x\ \min)^2]$

Get the histogram of the value array
This is the MESAH array
```

After plotting the "value" array in a histogram, one looks for isolated peaks, which are claimed to be indicators of chaotic behavior. The inspiration for this model appears to be a case of the logistic map: x(i+1) = 4x(i)(1-x(i)). In this map, one notices that "x" values near zero tend to be mapped to values near one, and vice-verse. The histogram for this case is below(figure 2); note the isolated peak at 0.5.

Figure 2

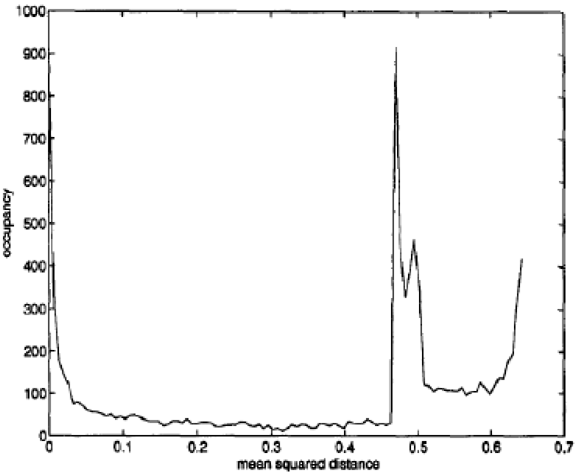
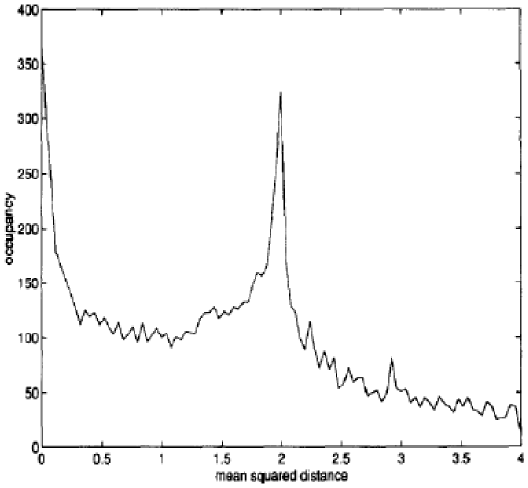


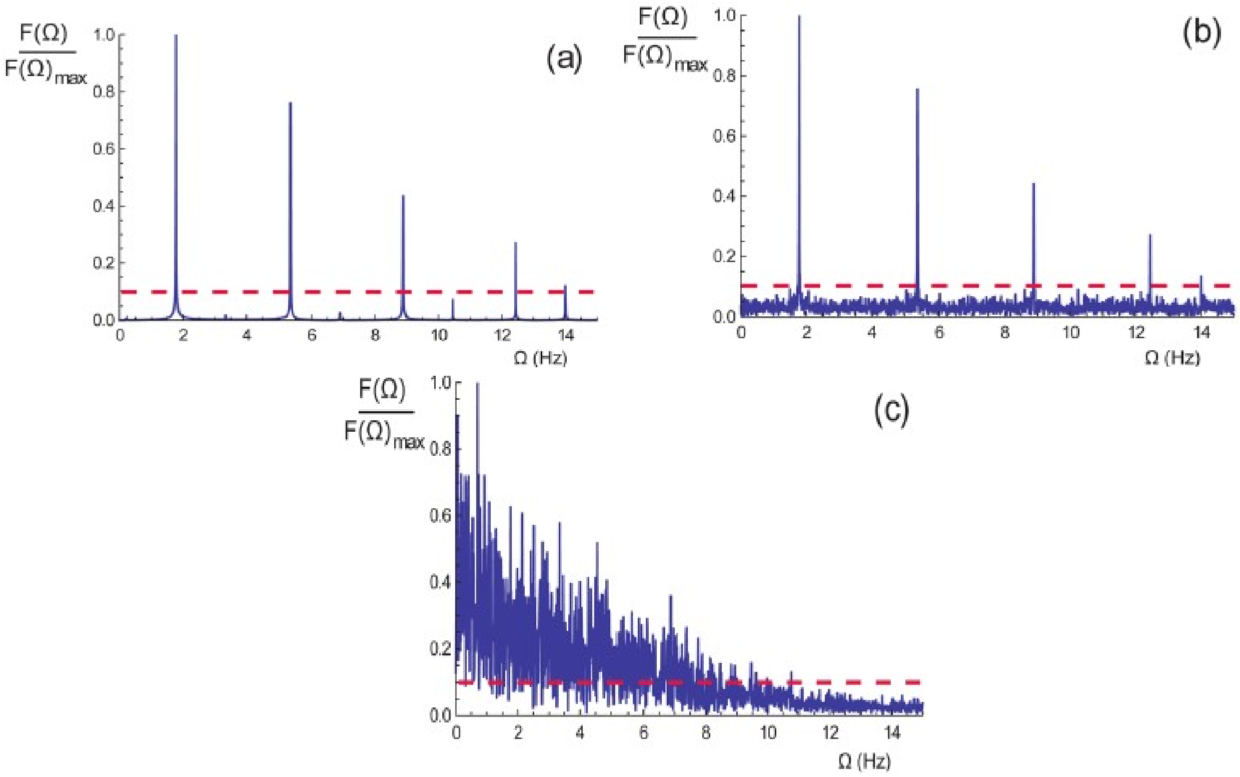
Figure 3



However, if we plot an identical distribution of data in a random order, we obtain figure 3 in which the peaks are much less isolated. McDonough also extends this model to higher dimensional cases like the Lozi map. In our work, we treat the iterates of embedded EKG data as a map from one state to another and construct the MESAH. We hope to attain qualitative information about the repeated patterns seen in such data. The addition of low levels of noise to the test problems for this detector did not detract significantly from it's effectiveness—white noise, for example, superimposes a normal distribution—making it an promising option to apply to observed data.

Our final heuristic chaotic indicator can be applied directly to the EKG data without an embedding. In this method, the discrete Fourier transform is applied directly to a subset of the data. The energy at various frequencies is then plotted in a spectrogram. In the case of truly periodic data, only a few local maxima will bee seen in this plot (see figure 4a). However, the addition of noise to the data, many additional maxima at a low amplitude (figure 4b). Alternatively, if the data is chaotic in nature, we tend to find many local maxima at large magnitudes(figure 4c). We can characterize the nature of a data set by the number of maxima present in the spectrogram above an empirically determined threshold (the dotted red line). In this way, we obtain a quantitative measurement about the system's periodic tendencies.

Figure 4



While Lyapunov exponents and dimensions have been calculated previously from EKG data, our work will differ in a few regards. Casaleggio and Braiotta applied only Rosenstein's algorithm to approximate the exponent, as implemented in "lyap_r," and they calculated only the lyapunov dimension as opposed to the correlation dimension. Owis et. al. applied Wolf's algorithm to calculate the Lyapunov exponent and Grassberger Procaccia to find the correlation dimension. However, we believe Wolf's algorithm to be less accurate than our methods, and Owis used small data sets (fewer than 2,000 points in each trial) which may explain their low "m" values. While this number of points appears acceptable for their dimension calculation, it brings suspicion to their exponent. On the other hand, Casaleggio always uses a data set of 50,000 points, which may be overly large in some cases. While this may allow one to obtain more false-nearest-neighbors, it is quite rare for someone to exhibit an arrhythmia of interest over a data set that large. Therefore it seems possible that their results could be contaminated by more regular beats for these trials. By choosing a sample size of 20,000 for our Lyapunov calculations, we hope to hone in more heavily on the areas of interest while still preserving credibility. However, in certain trials, we are unable to meet Ekman and Ruelle's recommendations due to a large calculated correlation dimension (it recommends hundreds of thousands of points). We acknowledge that some trials may lack accuracy because of this, but the raising the neighborhood size or number of points would reduce accuracy and diagnostic power in any event.

In addition, to our knowledge, the Fourier and MESAH methods proposed have not yet been applied to EKG data.

Results: We next apply the various methods above to data obtained from the MIT-BIH arrhythmia database. In these data sets, measurements were taken from a variety of patients exhibiting different cardiac phenomena. Voltages were measured at a rate of 360Hz, in integer values generally in a range around 890 to 1250. When the parameter is available, we apply a theiler window of 100 iterations in order to avoid using false-nearest-neighbors that are two similar to the primary point. Neighborhoods are determined with several different length scales ranging from fifteen to twenty-five units.

We tested our indicators on several 20,000 point subsets of the MIT-BIH arrhythmia data sets as listed in table 2.

Table 2: Approximate Values of Indicators

dataset	tau	m	Starting point	Cor. dimension	lyap_r	lyap_k	Fourier peaks	Behavior
100	39	14	1	2.8	0.36	0.35	25	Sinus
103	32	11	1	2.47	0.32	0.35	21	Sinus
105	21	13	67	2.6	0.15p	0.17p	11	Sinus (with a couple irregularities)
205	43	12	525782	2.73	0.15	0.16	4	V. Tach/sinus
207	93	11	13868	3.1	0.05	failed	4	V. Tach/Flutter
207	135	13	101185	3.4	0.04	failed	1	Blocked branch but sinus rhythm
213	31	12	53845	3.98	0.12	0.2	5	V. Bigeminy/sinus
213	30	12	115686	3.4	0.12	0.2	8	Sinus
213	30	12	167075	3.3	0.11	failed	6	V. Bigeminy/sinus
223	34	13	1	2.6	0.18	0.15	12	Sinus
223	42	13	43517	2.6	0.2p	0.15p	10	V. Tigeminy
223	39	13	298391	2.96	0.2p	0.21	12	V. Bigeminy
223	38	12	519178	2.8	0.3p	0.17p	6	V. Tach.

Note that all of the files had been annotated to indicate when cardiac events of interest occur. The plots from "lyap_r" and "lyak_k" seen below in figures 5 and 6 are typical in quality of the output from all files examined. While we did implement an algorithm to attempt an automatic linear regression, if the best fit line did not appear representative of the plotted points, it was disregarded and recalculated by hand. This introduces some measure of uncertainty to our measurements that is difficult to quantify. The algorithms were performed for five values of neighborhood sizes between fifteen and twenty-five. They were also run for two different values of "m"; once at the recommended embedding dimension found with "false_nearest" and again at "m+1." The letter "p" was added next to elements in table 2 if these two quantities differed by more than ten percent, indicating that they have questionable validity. It appears that "lyap_r" is a bit more robust than "lyap_k", as the latter failed to find sufficient false-nearest-neighbors in several cases. Output from both algorithms had largely nonlinear structure, prompting us to truncate the data set when attempting our linear regression. "Lyap_k" in particular uniformly produced wildly unhelpful iterates for well over half of the output.

Despite this, some worthwhile results were obtained; we tend to support Casaleggio and Braiotta's conclusion that normal sinus rhythm (as seen in files 100, and 103) tended to produce larger Lyapunov exponents than the more irregular files. In particular, exponents for ventricular bigeminy and trigeminy had larger exponents than ventricular tachycardia and flutter. Note that while files 213 and 223 have entries labelled as sinus rhythm. These were also patients who experienced irregular cardiac behavior in the same thirty minute window, and even if the rhythm is considered "normal", the beats frequently weren't. In addition, the correlation dimension of several files exceeded 3.0. From Eckman and Ruelle, we know this suggests they may require hundreds of thousands of points to confidently compute the exponent.

Figure 5

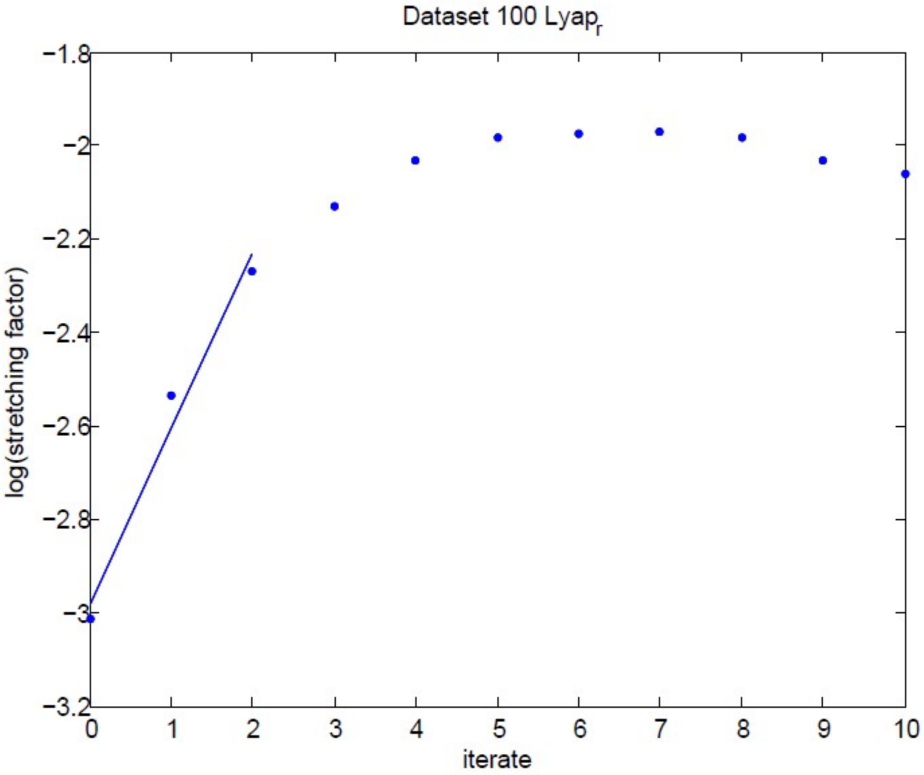
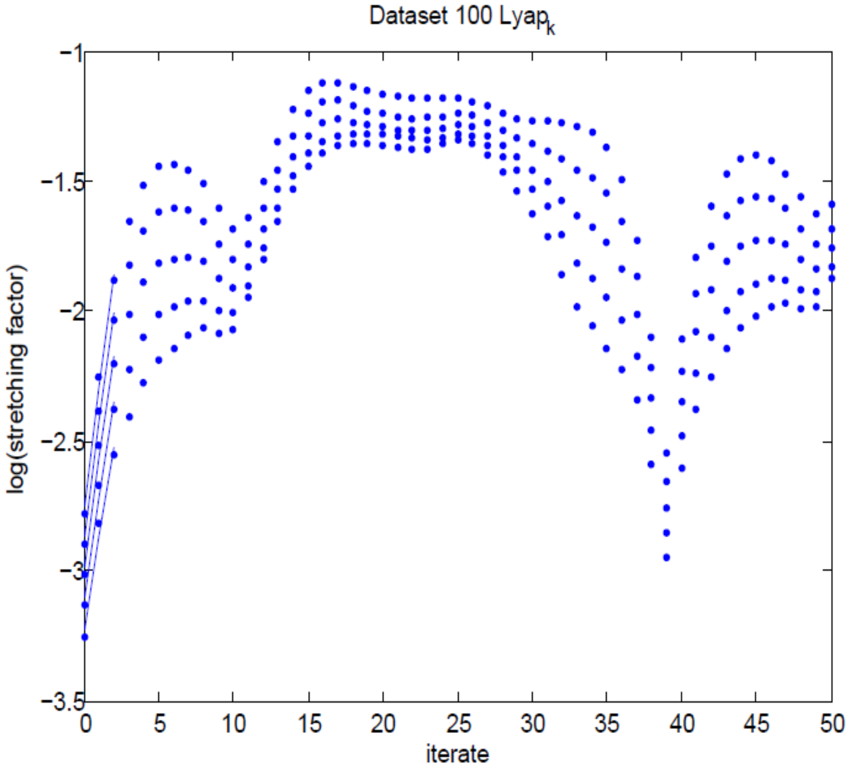


Figure 6



We failed to extract any useful information possibly contained from the mean squared error histograms. Figure 7 shows a histogram from (presumably chaotic) regular sinus rhythm on the left and a much more complicated pattern on the right. We saw none of the isolated peaks that would indicate chaos in any histograms generated; most appeared to be some variation on a normal bell curve. While the normal curve is the expected output of white noise, I believe that this results is more likely due to a failure of the method in this application.

Figure 7a

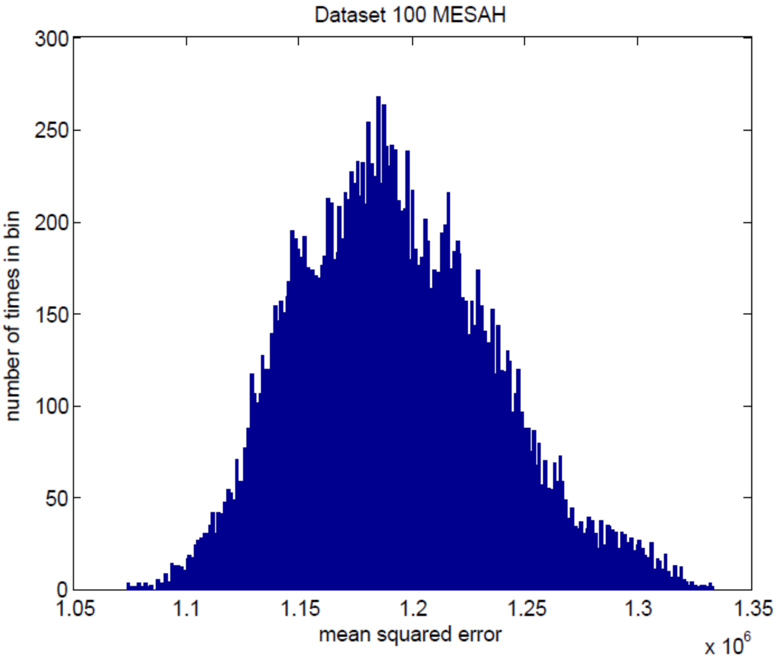
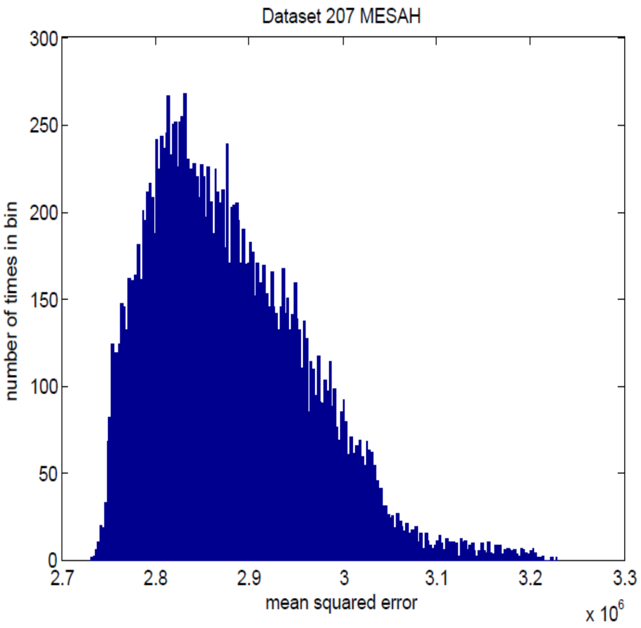
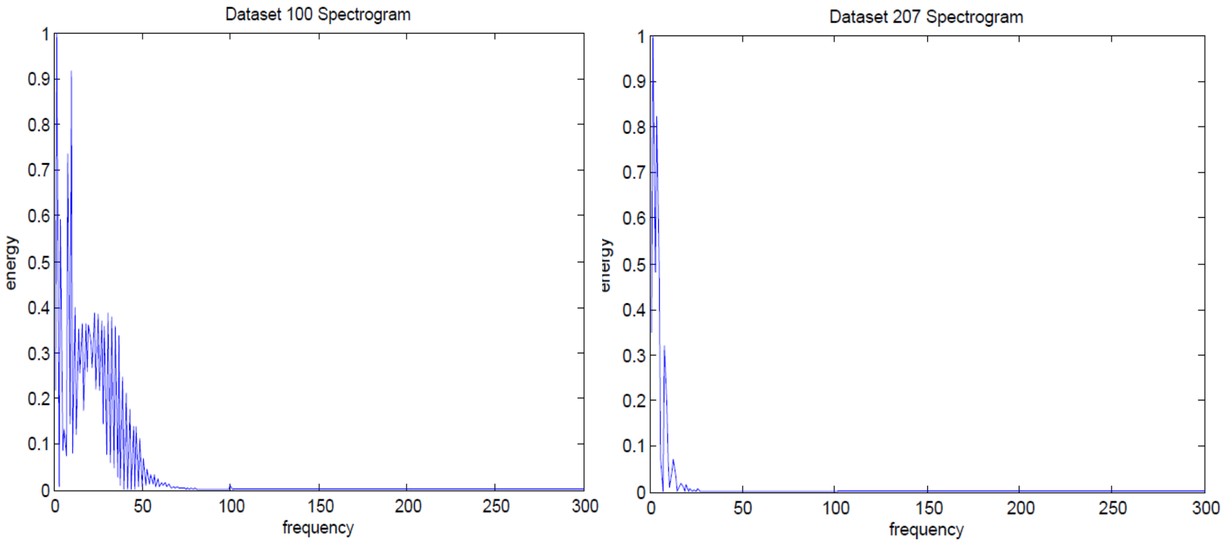


Figure 7b



The spectrograms, however, allowed us to compute a very useful metric of chaos that strongly supports the Lyapunov exponent results. A higher number of local maxima above the threshold of ten percent were counted in the trials with normal sinus rhthym than those with more irregular behavior. See figure 8 for examples from sinus rhthym and ventricular tachycardia.

Figure 8



Conclusion: In this paper, we applied several methods of chaos detection to EKG data. Choosing the appropriate number of points to apply our methods to is a nontrivial task, as we seek to obtain accuracy in our calculations while capturing dynamics that occur over a short time. Clearly some rounding has been applied to the measurements, and we should expect our results to indicate this fact. Indeed, the presence of any white noise at all means that the true greatest Lyapunov exponent is quite large. "Lyap_r" more regularly produced results than "lyap_k", but even this algorithm failed to be robust across different values of "m" in some cases. Our reliance on ad hoc linear regression methods is also somewhat disconcerting. Nonetheless, we were able to reproduce most of the results seen by Casaleggio and Braiotta; sinus rhythm appears to produce larger Lyapunov exponents than ventricular bigeminy and ventricular tachycardia.

The MESAH method revealed little about the system, and I must question its validity outside of low dimensional systems in its present form.

The application of spectrograms to the EKG data was quite successful. In addition to being computationally cheap, the metric obtained by counting peaks above a threshold coincided with results from Lyapunov exponents, suggesting that sinus rhthym is "more chaotic" than the others.

Future studies could add denoising algorithms, a standardized method of linear regression, and perhaps a higher resolution of the lower frequency Fourier modes. Data obtained at a higher frequency could possibly allow one to more accurately assess the Lyapunov exponents of different arrythmia, though oversampling may be a problem.

References:

Casaleggio, A and Stefano Braiotta. “Estimation of Lyapunov Exponents of ECG Time Series—The Influece of Parameters.” Chaos, Solitons, and Fractals 8:10 1591-1599, 1997

J. Eckmann and D. Ruelle. “Fundamental Limitations for Estimating Dimensions and Lyapunov Exponents in Dynamical Systems.” Physica D: Nonlinear Phenomena 56:185-187. 1992.

Gothwal, H., Kedawat, S., and R. Kumar. "Cardiac Arrhythmias Detection in an ECG Beat Signal Using Fast Fourier Transform and Artificial Neural Network." J. Biomedical Science and Engineering, 4:289-296, 2011.

P. Mcdonough, J. Noonan, G. Hall. “A New Chaos Detector.” Computers & Engineering 21.6:417-431. 1995.

M. Owis, A., Abou-Zied, A. Youssef, and Y. Kadah. “Study of Features Based on Nonlinear Dynamical Modeling in ECG Arrhythmia Detection and Classification.” IEE Transactions on Biomedical Engineering. 49:733-735, 2002.

http://www.physionet.org/physiobank/database/mitdb/

Wiebe, R. And L.N. Virgin. “A Heuristic Method for Identifying Chaos from Frequency Content.” Chaos 22:013136

A Survey of Dynamical Systems Models for Language Change

Erik Silkensen

CSCI 5446, Spring 2012

Abstract

This report surveys dynamical systems models of language change, and discusses an attempt at applying one of the models to empirical data.

1 Introduction

Language is dynamic; it is always changing. Historical linguists, of course, have long been interested in studying and documenting the ways languages change. However, with the advent of computers, researchers have also recently become interested in studying the evolution of language from a computational perspective. Over the last twenty years, several researchers have proposed mathematical models of language change that are based on dynamical systems. This report surveys two of these models in particular, and discusses an attempt of the author's to apply one of the models to empirical data.

2 Language Change

This section briefly introduces two concrete examples of changes to the English language, to both its phonology and its morphology; afterwards, the report focuses on how one might model such changes mathematically.

2.1 Phonological

According to Trask [11], Otto Jespersen first studied the so-called *Great Vowel Shift* of the English language, a gradual change in the pronunciation of English vowels that took place throughout the 15th century. The following notation describes some of these changes using the International Phonetic Alphabet (IPA).

[iː] → [aɪ]
[aː] → [eɪ]
[ɛː] → [iː]
⋮ ⋮

The above rules indicate that the two highest, long vowels, [iː] and [aː] (the marker ː means 'long'), were *diphthongized*; meanwhile, the other long, open vowels, such as [ɛː], became more *closed*.

2.2 Morphological

Crowley [1] cites Wilhelm von Humboldt, a 19th century linguist, as the first to suggest that the morphological type of language would tend to oscillate between 'isolating' and 'synthetic', or 'agglutinative'—the *cycle of agglutination*.

Many languages have completed up to half of the hypothesized cycle. For example, Old English was a *synthetic* language: words were often inflected for their grammatical category. In contrast, Modern English has become a much more *isolating* language: word order determines grammatical category more often than inflection. Consider the following data [12]:

se-cyning	ofslog	þa-cyningan	*Old English*
NOM-king	kill-PAST	ACC-queen	*Morphology*
'the king killed the queen'			*Modern English*

The top line gives an Old English translation of the Modern English in the bottom line. The middle line annotates some of the morphemes in the Old English sentence: the *se* morpheme is a nominative case marker—it indicates that *king* is the subject of the sentence; *þa* is an accusative case marker—it says that *queen* is the object. As a result, word order in Old English can be free:

þa-cyningan	ofslog	se-cyning	*Old English*
ACC-queen	kill-PAST	NOM-king	*Morphology*
'the king killed the queen'			*Modern English*

The inflected nouns may appear either before or after the verb in Old English without changing the meaning of the sentence. Modern English has lost these case markers through its transformation into a more isolating language.

3 Models of Language Change

Linguists hope to model phonological and morphological (among other) kinds of language change, such as those in the previous section, in an effort to come up with both a descriptive and possibly predictive tool for their study of language. This section presents two recent models based on dynamical systems.

3.1 The Language Dynamical Equation

Nowak et al. [9] propose a model, called the *language dynamical equation*, inspired by the idea that language change can be explained by human learning error. That is, suppose there is some finite set of languages learnable by the human brain, the 'Universal Grammar' $UG = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$, where $\mathcal{G}_i$ is some language, or grammar, in the set. Then one can think of language change as the result of an imperfect learning process in which a child acquires language $\mathcal{G}_j$ after being exposed to sentences from their parent's language $\mathcal{G}_i$.

Given a population that speaks n different languages, let x_i be the fraction of the population that speaks language $\mathcal{G}_i$ such that $\sum_i x_i = 1$. The language dynamical equation computes $\dot{x_i}$, the change in that fraction of the population, for each language. Specifically, the equation is

$$\dot{x_j} = \sum_{i=1}^{n} \mathcal{F}_i x_i \mathcal{Q}_{ij} - \phi x_j$$

where $\mathcal{F}_i$ is the *fitness* of language $\mathcal{G}_i$, ϕ is the average fitness across languages, and $\mathcal{Q}_{ij}$ is the probability that a child will acquire language $\mathcal{G}_j$ given that their parent speaks $\mathcal{G}_i$. The fitness of a language $\mathcal{G}_j$ is computed in the following way,

$$\mathcal{F}_j = \sum_{k=1}^{n} \mathcal{B}_{jk} x_k$$

where $\mathcal{B}$ is a pay-off matrix that gives the *benefit* at each $\mathcal{B}_{ij}$ to a speaker of $\mathcal{G}_i$ of a meeting with a speaker of $\mathcal{G}_j$.

Mitchener and Nowak [7] analyze instances of their equation that lead to limit cycles and chaos, and argue that therefore "simple learning errors can lead to complex, unpredictable, and seeming stochastic changes in languages over time." The remainder of this section presents those instances of the model.

Example 3.1. Consider a population of three languages, and let the pay-off and learning matrices be the following:

$$\mathcal{B} = \begin{pmatrix} 0.88 & 0.2 & 0.2 \\ 0.2 & 0.88 & 0.2 \\ 0.2 & 0.2 & 0.88 \end{pmatrix} \qquad \mathcal{Q} = \begin{pmatrix} 0.79 & 0.2 & 0.01 \\ 0.01 & 0.79 & 0.2 \\ 0.2 & 0.01 & 0.79 \end{pmatrix}$$

The diagonally dominant pay-off matrix $\mathcal{B}$ signifies that "all languages are equal" in this scenario; the imperfect learning matrix $\mathcal{Q}$ establishes a distinct first, second, and third best language for each possible pairing. Figure 1 shows the orbit generated by the language dynamical equation when instantiated with these matrices. Mitchener and Nowak claim that such a limit cycle is "suggestive of the morphology type cycle" hypothesized by von Humboldt, taking x_1, x_2, and x_3 to be different levels of agglutination, for example.

Example 3.2. Next, expand the population to five languages and use these matrices, with $\mathcal{Q}$ now parameterized over μ:

$$\mathcal{B} = \begin{pmatrix} 0.88 & 0.2 & 0.2 & 0 & 0.3 \\ 0.2 & 0.88 & 0.2 & 0 & 0.3 \\ 0.2 & 0.2 & 0.88 & 0 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.88 & 0 \\ 0 & 0 & 0 & 0.3 & 0.88 \end{pmatrix} \mathcal{Q} = \begin{pmatrix} 0.75 & 0.2 & 0.01 & 0.04 & 0 \\ 0.01 & 0.75 & 0.2 & 0.04 & 0 \\ 0.2 & 0.01 & 0.75 & 0.04 & 0 \\ 0 & 0 & 0 & \mu & 1-\mu \\ 1-\mu & 0 & 0 & 0 & \mu \end{pmatrix}$$

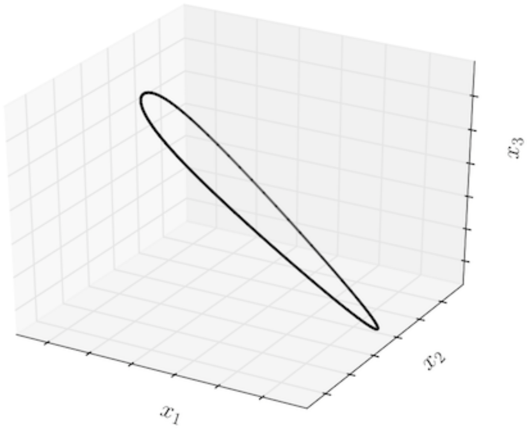


Figure 1: A limit cycle generated by the language dynamical equation that resembles the “cycle of agglutination” (see Example 3.1).

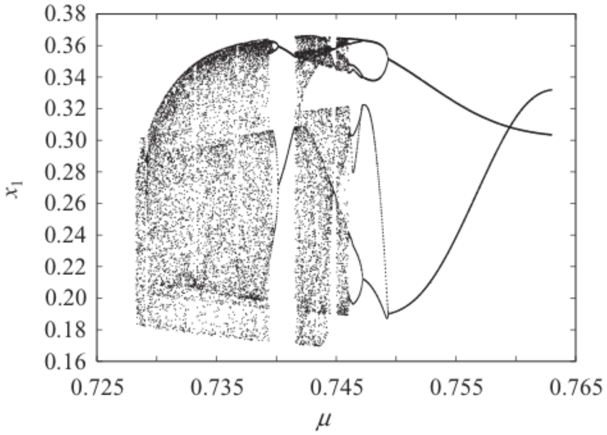


Figure 2: A bifurcation diagram of the language dynamical equation exhibiting chaos, e.g., at $\mu = 0.735$ (see Example 3.2).

In this scenario, μ configures the probabilities for the new fourth and fifth languages. Mitchener and Nowak carefully construct the matrices so that they can induce chaos. Figure 2 shows a bifurcation diagram for a range of μ values, with x_1 on the y-axis and μ along the x-axis. The plot features the kind of period-doubling bifurcations leading to chaos at, e.g., $\mu = 0.735$. Mitchener [6] gives a thorough bifurcation analysis of the language dynamical equation.

3.2 The $\mathcal{G}$–$\mathcal{A}$–$\mathcal{P}$ Model

Niyogi and Berwick [8] derive a parametric model of language change also at the population level from one of language acquisition at the individual level that is analogous to the logistic map. The parameters of their system are $\mathcal{G}$, now a *family* of grammars, $\mathcal{A}$, a learning algorithm that maps sets of sentences to a grammar $g \in \mathcal{G}$, and $\mathcal{P}$, a set of primary linguistic data presented to the children acquiring language. The $\mathcal{G}$–$\mathcal{A}$–$\mathcal{P}$ model essentially computes the changing linguistic population dynamics, similar to the language dynamical equation; a formal presentation of the model is left to the original paper.

Niyogi and Berwick applied their model to data exhibiting the loss of the *verb-second* (V2) phenomenon as the Old French language evolved into Modern French. Later, Sonderegger and Niyogi [10] updated and then applied the model to more empirical data in an analysis of stress shift in English noun/verb pairs. That is, they investigated pairs of *heteronyms*, words that are spelled the same but are either nouns or verbs depending on which syllable is stressed (e.g., ‘récord’ [noun] and ‘recórd’ [verb]). They looked at pronunciation guides in dictionaries throughout history to trace the changing patterns in stress. In the future, they hope to continue tuning their model's accuracy with historical data so that it might be able to generate predictions of change to language over time.

4 Evaluation

This section discusses possible avenues for an empirical evaluation of the language dynamical equation, in addition to raising a few questions about the applicability of the model.

4.1 Sources of Data

More language data is easily available on-line now than ever before. Google claim to have collected roughly 4% of all books *ever printed* in their enormous, freely accessible n-gram database. An n-gram is a sequence of n words; Google computes a probabilistic model that gives the likelihood of a given n-gram occurring in English text.

Michel et al. [5] describe the construction of the Google database and show several interesting plots of changes to English vocabulary over the last two hundred years. However, to apply a model, such as the language dynamical equation, in hopes of studying some sort of grammatical change to language (e.g., the loss of English case), to Google's data is difficult. Even if the data went far enough back into history, one would need to morphologically analyze the words, and also be comfortable programming with many gigabytes of data.

The CHILDES project [4] provides a database of child language transcripts, among several other tools. Although orders of magnitude smaller than Google, the CHILDES database includes full sentences complete with part-of-speech tags. While time ran out for this (i.e., this class report) project, it would be interesting to see how one might use the CHILDES database to study the dynamics of language change in the future.

4.2 Reflection

Mitchener and Nowak present a clean mathematical model in their language dynamical equation; however, the examples they chose for the article don't necessarily make the best argument for its applicability to real language. For example, they show how to induce chaos, but don't suggest what relevance chaos has to actual language change. They motivate their first example by citing the morphology type cycle, but there's no evidence of a language every completing more than half the cycle—it may not exist. Australian linguist Dixon[1] [2] has said that, should a language ever complete the hypothesized cycle of morphological type, it may take "probably anything from two or three thousand years to fifty thousand and more."

After reflection, it seems as if the language dynamical equation is probably mostly of theoretical interest. Sharing some of this sentiment, Kello [3] comments that "it is hard to see how future research will provide more specific evidence on whether real language change is driven by the principles of their model."

5 Conclusion

This report surveyed two dynamical systems models of language change. The first sometimes exhibits limit cycles and chaos. Unfortunately an effort to apply it to empirical data has so far been unsuccessful, and it's not certain that the model is not meant to be only of theoretical value. However, the Google n-gram and CHILDES databases seem like potentially good sources of data for an empirical investigation of a dynamical systems language model. The designers of the second surveyed model have used their model to study phonological and morphological changes to English and French, respectively.

[1] Possibly a colleague Dr. Bradley's at La Trobe University?

References

[1] T. Crowley. *An Introduction to Historical Linguistics.* Oxford University Press, 3rd edition, 1997.

[2] R. M. W. Dixon. *Ergativity.* Cambridge University Press, 1994.

[3] C. T. Kello. Characterizing the evolutionary dynamics of language. *Trends in Cognitive Sciences*, 8(9):392–394, 2004.

[4] B. MacWhinney. *The CHILDES Project: Tools for Analyzing Talk.* Lawrence Erlbaum Associates, Inc., 2000.

[5] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182, 2011.

[6] W. G. Mitchener. Bifurcation analysis of the fully symmetric language dynamical equation. *Journal of Mathematical Biology*, 46(3):265–285, 3 2003.

[7] W. G. Mitchener and M. A. Nowak. Chaos and language. *Proceedings of The Royal Society of London. Series B, Biological Sciences*, 271(1540): 701–704, 2004.

[8] P. Niyogi and R. Berwick. A dynamical systems model for language change. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1994.

[9] M. A. Nowak, N. L. Komarova, and P. Niyogi. Evolution of universal grammar. *Science*, 291(5501):114–118, 2001.

[10] M. Sonderegger and P. Niyogi. Combining data and mathematical models of language change. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1019–1029, Stroudsburg, PA, USA, 2010.

[11] R. L. Trask. *Historical Linguistics.* Arnold, London, 1996.

[12] A. Tålig. The decay of the case system in the English language. Technical report, Luleå University of Technology, Luleå, Sweden, 2008.

Variations in Indian Classical Percussion using Chaos

Yogesh Virkar[1], *

[1]*Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80305 USA*

(Dated: May 3, 2012)

In the past, chaos has been used successfully to produce variations in musical compositions. The property of sensitive dependence on initial conditions is exploited to make these variations. In this article, I demonstrate how to make variations in compositions of a percussion instrument in Indian classical music. My aim is to produce variations that respect the rules of this style of music. For this purpose, the composition is mapped on two different Lorenz attractors, which serve as two reference trajectories. A new initial condition is chosen and the mappings for the new trajectory are found using appropriate distance measure and the appropriate reference trajectory. These mappings found on the new trajectory correspond to a variation. The variations formed this way are shown not only to respect rules of Indian classical music but also to sound good.

I. INTRODUCTION

Chaos has numerous applications in various disciplines [7] such as physics, chemistry, engineering, etc. Interestingly enough, chaos has also been successfully used in the past to explore different forms of art such as music [2]. Since Dabby's work, chaos has been used to make chaotic variations in dance [1], designing rock-climbing routes [5], etc.

This work is influenced from Dabby's work on making musical variations using a chaotic Lorenz attractor. In Dabby's scheme, a sequence of musical pitches p_i are paired with the x components of a chaotic Lorenz trajectory. The idea is to use this as a reference trajectory. We then drop a new initial condition to trace out a new trajectory. Using some distance measure, the new trajectory can be compared with the reference trajectory and musical pitches are assigned to the new trajectory.

Even if the initial condition is chosen close to the reference trajectory, the sensitive dependence on initial conditions property of chaos gives rise to variations of the original mapped composition. One important aspect of this work is that depending on the initial condition chosen, we can have variations that maintain to some extent the same style as the original piece or ones that mutate the original piece beyond recognition. In either case, this approach forms the basis of this project.

*Electronic address: `yogesh.virkar@colorado.edu`

1	2	3	4	5	6	7	8
dha-ti	dha-ge	na-dha	tere-kita	dha-ti	dha-ge	ti-na	ki-na

FIG. 1: *Kaida*, the basic composition. The phrases or *words* are written in the English Alphabet but they are phonetically equivalent. For each phrase, parts on either side of the hyphen indicate strokes used to make that phrase. They are numbered from 1 to 8 for ease of understanding variations shown latter.

In Indian classical music compositions are played within a fixed number of beats which form a rhythmic cycle called *Taal* [4]. Variations are built from a basic composition in any *Taal*. For a single variation, the rules of Indian classical music allow variability during the start of this rhythmic cycle or *Taal* where the musician improvises but as we progress temporally the rule states that the basic composition should be played in whole or in part.

Unlike Dabby's scheme, we use two reference trajectories. One is the chaotic Lorenz attractor and the second is a fixed point attracting setting of the Lorenz attractor. We switch from the chaotic map to the fixed point attracting map as we progress temporally. The chaotic attractor captures the variability during the start of a *Taal*, while the fixed point attractor captures the aspect of playing the basic composition in whole or in part. This technique is shown to be effective and results are provided using a simple basic composition in a *Taal* of 16 beats. Some variations sound close to the ones played in actual recitals, while some seem particularly unconventional.

II. INDIAN CLASSICAL MUSIC

The theory and basics of Indian classical music pertaining to percussion, i.e. the Tabla, has been discussed extensively in [3]. We review some of the basics required for our discussion. I will also introduce the basic composition that I used for this work.

The Tabla is composed of two drums, the right hand drum which gives sharp sounds and the left hand drum which gives the bass effects. Different strokes can be played on each of these drums. The combination of the strokes can be used to produce a rhythmic cycle of some fixed number of beats called as the *Taal*. Certain combinations of these strokes can also give rise to *words*, which are the building blocks of a basic composition, also called as *Kaida* (meaning the "rule"). *Words* are immutable in the sense that a word cannot be played in part. Each word usually spans 1 beat. Compositions are played in a specific *Taal* by making variations of the *Kaida*.

The classical style of playing the Tabla allows one to make improvisations or variations on this predefined "rule", but enforces the following constraints

1. The improvisations made should fit in the *Taal*, in the sense that they should start and finish within fixed number of rhythmic cycles.

2. The improvisations should use permutations or combinations of the different *words* defining the *Kaida*.

3. As one improvises and moves away from the *Kaida* during the start of the rhythmic cycle, one must play the *Kaida*, wholly or in part, towards the end of the rhythmic cycle.

The *Taal* used in this project is called *Tintaal* which is composed of 16 beats. The variations made in this project span two cycles of the *Tintaal* i.e. 32 beats. The *Kaida* used in this project consists of 8 immutable phrases or *words* each spanning 1 beat in time. These phrases are numbered from 1 to 8 in the order they appear in the *Kaida*. This is shown in Figure 1. We will refer to these phrases using the notation $p_1, p_2, ..., p_8$ or equivalently p_i for the i^{th} phrase in the basic composition.

III. METHODOLOGY

Rules 1 and 2 described in the previous section are trivial for a computer program. For rule 3 I propose the following methodology. It consists of four main aspects. Section III A deals with exploring the musical space or making variations during the start of the *Taal*. Section III B deals with how mappings are done. Section III C shows how to incorporate the rule 3. Section III D handles the issue of a same phrase repeated for multiple successive beats.

A. Using Chaos to make variations

Similar to Dabby's scheme, we form a mapping between the *Kaida* and the trajectory formed using fourth-order Runge-Kutta solver (RK4) on a chaotic setting of the Lorenz attractor with parameters $a = 16$, $r = 45$ and $b = 4$. The Lorenz system is defined by the following ordinary differential equations,

$$\dot{x} = a(y - x) \tag{1}$$

$$\dot{y} = rx - y - xz \tag{2}$$

$$\dot{z} = xy - bz \tag{3}$$

This reference trajectory (shown in black in Figure 3) was formed with initial condition $[x_0, y_0, z_0] = [1, 0, 0]$. Each phrase p_i is mapped on the projection of the Lorenz attractor on the xz-plane. Thus each (x, z) in the reference trajectory is associated with one of the elements in p_i. Thus $(x_1, y_1) \rightarrow p_1$, $(x_2, y_2) \rightarrow p_2$, $(x_3, y_3) \rightarrow p_3$ and so on and so forth. Note that $(x_9, y_9) \rightarrow p_1$ as the phrases wrap around. This is slightly different from Dabby's scheme where only x-coordinates are used.

B. Using a distance measure to produce mappings

The next step is to select a new initial condition and run RK4 solver with the same chaotic setting. Let this new trajectory be $[x', z']$. The mapping for this trajectory are found using the reference trajectory of the previous section. To find mapping for $[x'_1, z'_1]$, we find the point in the reference trajectory $[x, z]$ that is closest to $[x'_1, z'_1]$. Thus we find the point $[x_i, z_i]$ such that,

$$\forall [x_i, z_i] \in [x, z], [x_i, z_i] : \min_i \left(\sqrt{(x_i - x'_1)^2 + (z_i - z'_1)^2} \right) \tag{4}$$

Thus we use the euclidean distance as a distance measure. If the Equation (4) gives $[x_{10}, y_{10}]$ as the closest point then according to our mapping scheme $[x'_1, z'_1] \rightarrow p_2$, i.e., $[x'_1, z'_1]$ maps to phrase 2.

If we use a chaotic map as our reference and use chaotic setting of Lorenz's equations given by (1), (2) and (3), then choosing a slightly different initial condition we get variability and improvisations of the *Kaida*. This is due to the sensitive dependence on initial condition property of chaos.

C. Switching between chaotic and fixed point attractor probabilistically

However, we want to follow the *Kaida* in whole or in part towards the end of the *Taal*. For this we use Lorenz system with parameters $a = 16$, $r = 23$ and $b = 4$. This setting gives us two fixed point attractors, one at $[x, z] = [-9.381, 22]$ and the other at $[x, z] = [9.381, 22]$. For this attracting reference map we have two reference trajectories (shown in red in Figure 3), one with initial condition $[1, 0, 0]$ and the other with $[-1, 0, 0]$. We map the phrases in the same way on both these trajectories.

Rule 3 states that as we progress temporally we should play the basic composition in whole or in part. For this we make a probabilistic switch of the reference maps and settings of Lorenz equations from the chaotic setting of Section III A to the fixed point attracting one mentioned above. This probability of switching is given as

$$\Pr(chaotic \to fixed\ pt.) = i/n \tag{5}$$

where i is the number of the current beat and n denotes the total number of beats. Thus probability of switching increases smoothly over time. The reason to make the switch probabilistic is so that it models to some extent the human spontaneity in playing a variation.

D. Avoiding consecutive repetitions of the same phrase

With the scheme mentioned above, a phrase might get repeated for consecutive beats or multiple successive beats. Though there is no rule which states that consecutive repetitions are not allowed, if the same phrase is repeated consecutively for 4 or 5 times, the variation does not sound good.

To handle this, we introduce one more probability in our framework. We can call this probability as the probability of repeating $(j-1)^{th}$ phrase for j^{th} beat. Empirically, a nice choice of for this probability came out to be $p = 0.05$, i.e., we allow consecutive beats to have the same phrase 5% of the time. However, this is a free parameter in our model that can be adjusted as one sees fit.

IV. RESULTS

With the simple recipe of Section III proposed to make variations, we turn to some of the results. Using the basic composition of Figure 1 and our reference maps, we select two different initial conditions. For the first variation, we choose $[x_0, y_0, z_0] = [1.02, 0, 0.01]$. For the second variation, $[x_0, y_0, z_0] = [26, 0, 30]$.

The results are shown in Figure 2. For the first variation, we see that the first 16 beats show variability while the last 16 beats show that basic composition is played entirely as numbers $1 - 8$ appear in order for the last 16 beats. This shows that the switching scheme indeed works and all rules of classical music are followed. *Variation1.wav* shows that the variation sounds good and pretty conventional as well. Variation 2 is slightly unconventional, but looking at just the numbers

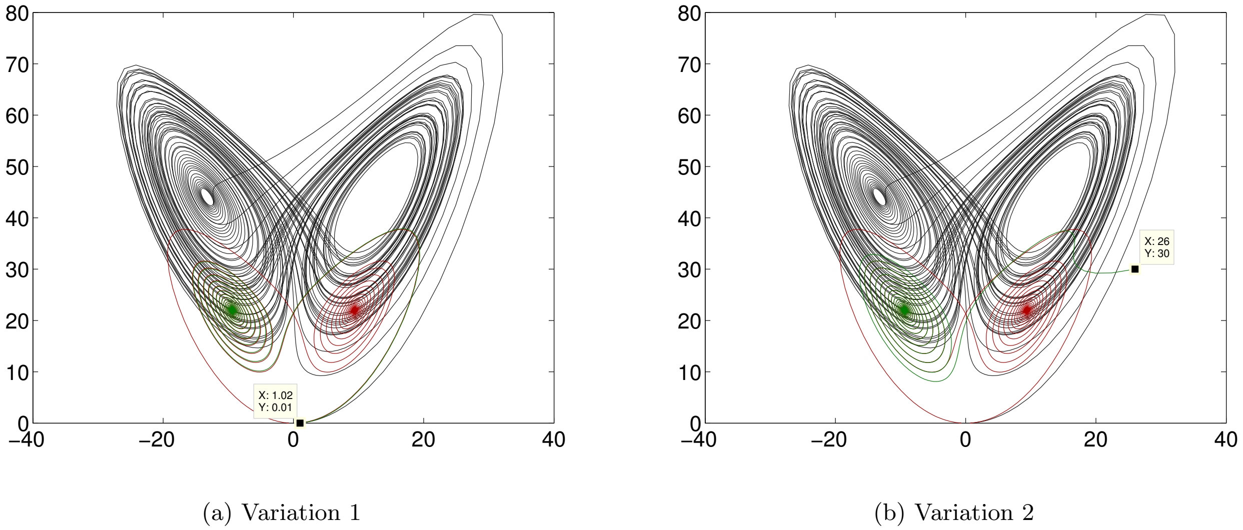
1	2	3	4	5	6	7	8
(1)	(2)	(4)	(2)	(5)	(6)	(7)	(8)
dha-ti	dha-ge	tere-kita	dha-ge	dha-ti	dha-ge	ti-na	ki-na
9	10	11	12	13	14	15	16
(1)	(8)	(2)	(3)	(5)	(6)	(7)	(8)
dha-ti	ki-na	dha-ge	na-dha	dha-ti	dha-ge	ti-na	ki-na
17	18	19	20	21	22	23	24
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
dha-ti	dha-ge	na-dha	tere-kita	dha-ti	dha-ge	ti-na	ki-na
25	26	27	28	29	30	31	32
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
dha-ti	dha-ge	na-dha	tere-kita	dha-ti	dha-ge	ti-na	ki-na

(a) Variation 1

1	2	3	4	5	6	7	8
(7)	(2)	(3)	(5)	(7)	(1)	(6)	(1)
ti-na	dha-ge	na-dha	dha-ti	ti-na	dha-ti	dha-ge	dha-ti
9	10	11	12	13	14	15	16
(4)	(5)	(4)	(6)	(5)	(7)	(8)	(1)
tere-kita	dha-ti	tere-kita	dha-ge	dha-ti	ti-na	ki-na	dha-ti
17	18	19	20	21	22	23	24
(2)	(1)	(3)	(4)	(5)	(6)	(7)	(8)
dha-ge	dha-ti	na-dha	tere-kita	dha-ti	dha-ge	ti-na	ki-na
25	26	27	28	29	30	31	32
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
dha-ti	dha-ge	na-dha	tere-kita	dha-ti	dha-ge	ti-na	ki-na

(b) Variation 2

FIG. 2: The numbers 1 to 32 denote the number of the beat. The numbers in brackets appearing below the beat number denote the number i of phrase p_i mapped to that beat. The numbers in red show that the basic composition was followed in whole or in part. The actual corresponding phrases appear in the English Alphabet below these numbers.



(a) Variation 1

(b) Variation 2

FIG. 3: Shows the variation trajectories in green. The chaotic Lorenz map, i.e., the first reference trajectory is shown in black. The fixed point attracting Lorenz map, i.e., the second reference trajectory is shown in red.

we see that from the 19^{th} beat, the basic composition is followed in part. Variation2.wav sounds good, though different than what one hears in a recital.

Figure 3 shows the two reference maps (chaotic in black and fixed pt. one in red) and the variation trajectory in green. From Figure 3(b) we can see how the switching takes place and we move from chaos to order as we return to the basic composition. This is less clear from Figure 3(a).

While these are good examples of variations, there are some variations with different ICs which are not good. Typically we might switch to the fixed point map late and depending on our IC we might not play the basic composition. A good prescription for an IC to avoid this is to start close to the IC of the reference trajectories.

But with certain combinations of ICs and the time of switching, we may not end up on p_8, i.e., the 8^{th} phrase for the 32^{nd} beat and such variations may not sound good. This is the problem of misaligned phrases. In this case, one option is force the last few beats to have phrases exactly in order so that play p_8 for the 32^{nd} beat. However, this kills the spontaneity and our model has more things to remember and hence we do not use such a scheme.

V. CONCLUSIONS

The primary aim of this project was to produce variations in Tabla that follow all the rules of Indian Classical music. While some of the rules are trivial and easy for a computer program to follow, others are not. The most important rule in playing variations of a composition was that the variation can explore the possibilities during initial beats of a Taal but as we progress temporally, the basic composition should be played in whole or in part. We used chaos to for exploring these possibilities initially and switched to a fixed point attractor setting to play the basic composition. This idea is inspired from the way musicians play this kind of music.

However there are some potential issues such as misalignment of phrases. One of the ways in which this can be handled is to force an alignment when the map is switched. However this does not model the spontaneity of a musical. Another viable option is to have a unstable periodic orbit and target the variation trajectory [6] towards it as we switch from the chaotic map. Thus instead of a fixed point attracting map, we would have a UPO as a reference trajectory. These are potential avenues for future research which could be worth exploring.

[1] E. Bradley and J. Stuart, *Using chaos to generate choreographic variations.*, Chaos **8** (1998), no. 800-807.

[2] D. Dabby, *Musical variations from a chaotic mapping*, Arts, Humanities, Social Sciences and Entrepreneurship **6** (1996).

[3] G. Farrell, *Thinking, saying, playing: Children learning tabla*, Bulletin of the Council for Research in Music Education (1997), no. 133, 14–19.

[4] O. Gillet and G. Richard, *Automatic labelling of tabla signals*, In Proceedings of International Conference on Music Information Retrieval.

[5] C. Phillips, L. Becker, and E. Bradley, *Strange beta: An assistance system for indoor rock climbing route setting using chaotic variations and machine learning*, American Institute of Physics **22** (2012).

[6] T. Shinbrot, E. Ott, C. Grebogi, and J. A. Yorke, *Using chaos to direct trajectories to targets*, Phy. Rev. Letters **65** (1990), no. 26, 3215–3218.

[7] S. Strogatz, *Nonlinear dynamics and chaos*, Perseus Books Publishing, LLC, 1994.

ANALYSIS OF CHAOTIC CELLULAR AUTOMATA FOR USE WITH SYMMETRIC KEY CRYPTOGRAPHY

Sean Wiese

April 30, 2012

1 Introduction

Over the past few decades the field of cryptography has become an increasingly important part of modern life. Tools that were once only employed by governments to protect state secrets are now used to protect everything from personal financial data to email logins. In the field of cryptography old encryptions systems are broken as new code breaking techniques are developed and computer performance increases. Consequently, Cryptographers are continuously looking for new algorithms to use in encryption systems.

One of the backbones of many major cryptography systems is a pseudorandom number generator (PRNG). These functions work to create a deterministic output that appears random based on a specific seed. A good PRNG is a deterministic mathematical function with high sensitivity to initial conditions, through key dependence, and an output that appears random. The field of Chaos Theory stresses a high sensitivity to initial conditions with outputs that can appear to be random, making chaotic functions an excellent candidate for a PRNG. A popular chaotic function, the logistic equation, has already been shown to be an effective PRNG[5]. A set of discrete dynamical systems called Cellular Automata (CA) have also been shown to be an effective PRGN for use in encryption systems [3][4]. Although the use of CAs in cryptography is not new, this paper seeks to compare the effectiveness of elementary cellular automata and life like cellular automata for use in encryption, as well as, the difference between a CA based stream cipher and CA based block cipher.

In Section 2, we will discuss the various types of cellular automata used in our encryption systems. In Section 3, a stream cipher based on an elementary cellular automaton will be described and analyzed. In Section 4, a stream cipher based on a 2 dimensional life like cellular automaton will be described and analyzed. In Section 5, a block cipher encryption system based on a 2 dimensional life like cellular automaton will be proposed and tested in detail. Finally, in Section 6, the paper will end with a comparison of the advantages and disadvantages of the three encryption systems, possible weaknesses of CA based encryption, and potential future extensions.

2 Cellular Automata

In this section we will provide a definition of a cellular automaton and describe the two different types of CAs that will be used in this paper.

2.1 CA Definition

A cellular automaton is a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling. It consists of a regular grid of cells, each in one of a finite number of states, such as ”On” and ”Off”. The grid can be in any finite number of dimensions. For each cell we will refer to the surrounding cells in the grid as neighbor cells. These neighbor cells can be defined to be any discrete distance away from the center cell. An initial state is selected by assigning a state for each cell. A new generation is created, according to some fixed rule that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Typically, the rule for updating the state of cells is the same for each cell and does not change over time.

2.2 Elementary 1D CA

Elementary 1D CAs are the simplest form of cellular automata. They exist in only two states, On or Off, and the rules that govern the system only depend on the nearest neighbor values. This gives a possible 2^3 possibly binary states, thus, producing 2^8 possibly 1D elementary CAs each of which can be represented by an 8-bit binary number. These cellular automata were extensively studied and classified by Stephen Wolfram. Stephen classified elementary cellular automata into four different classifications. [8]

- class 1: Nearly all initial patterns evolve quickly into a stable, homogeneous state. Any randomness in the initial pattern disappears.
- class 2: Nearly all initial patterns evolve quickly into stable or oscillating structures. Some of the randomness in the initial pattern may filter out, but some remains. Local changes to the initial pattern tend to remain local.
- class 3 : Nearly all initial patterns evolve in a pseudo-random or chaotic manner. Any stable structures that appear are quickly destroyed by the surrounding noise. Local changes to the initial pattern tend to spread indefinitely.
- class 4 : Nearly all initial patterns evolve into structures that interact in complex and interesting ways. Class 2 type stable or oscillating structures may be the eventual out-

come, but the number of steps required to reach this state may be very large, even when the initial pattern is relatively simple. Local changes to the initial pattern may spread indefinitely.

For purpose of this paper we will only focus on class 3 CAs.

2.3 2D Life Like Cellular Automata

A 2D life like CA differs from elementary cellular automata in that it exists in a 2 dimensional grid and must meet the following criteria:

- The array of cells of the automaton has two dimensions.
- Each cell of the automaton has two states (conventionally referred to as ”alive” and ”dead”, or alternatively ”on” and ”off”)
- The neighborhood of each cell consists of the eight adjacent cells and (possibly) the cell itself.
- In each time step of the automaton, the new state of a cell can be expressed as a function of the number of adjacent cells that are in the alive state and of the cell’s own state.

This class of cellular automata is named for the Conways Game of Life, the most famous cellular automaton, which meets all of these criteria.

Throughout this paper we will be using B/S notation to define CAs. This notation system indicates the number of neighbors that can result in a Birth, a change of a dead cell to an alive cell, and the number of neighbors that can result in a Survival, an alive cell remaining alive. For example, Conways Game of Life is denoted by (B3,S23) indicating that 3 neighbors allows a cell to be born and 2 or 3 neighboring cells allows the cell to survive.

3 1D CA Stream Cipher

In this section the development of an encryption system using a chaotic elementary CA will be discussed.

3.1 Stream Cipher

In cryptography, a stream cipher is a symmetric key cipher where plaintext bits are combined with a pseudorandom cipher bits. In a stream cipher each plaintext bit is encrypted one at a time with the corresponding bit of the cipher stream, to give a bit of the ciphertext stream. The XOR operation is used to combine a plaintext bit and a cipher bit into the ciphertext bit.

3.2 Choice of 1 Dimensional CA

A good stream cipher will have the following characteristics.

1. Apparent Randomness : The stream must appear to be random. If a psudoranom stream is XORed with a non-random stream the output will appear random.

2. Diffusion : A small change in the key used to generate the stream should provide a completely different result, i.e. half of the bits should be flipped.

A chaotic system satisfies the first condition. Therefore, we will look to select one of the chaotic elementary cellular automata identified by Wolfram. Wolfram identified rules 22, 30, 126, 150, and 182 as chaotic cellular automata [8]. For this paper we will use Rule 30 as the basis of our stream cipher. This rule was chosen for having the unique characteristic that the center column of the rule produces a statistically random stream of bits with a high sensitivity to initial conditions. Furthermore, this CA is currently used in the computational software program Mathematica as a large number random number generator [7].

3.3 Algorithm

To encrypt a data stream using this method the following technique can be employed.

1. Set the key as the initial condition of the CA.

2. Run the CA for n iterations where n is the number of bits in the key. The goal of this step is to increase diffusion for bit flip in an edge bit of the key. For example, if the initial condition starts with the 16 bit key 1000000100000001 and then changes to 0000000100000001 the first 8 bits of the center column will be the same.

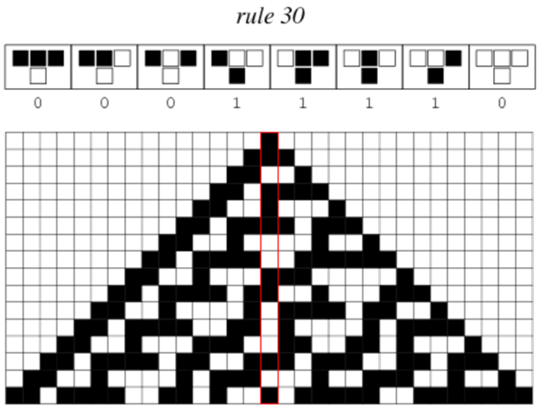
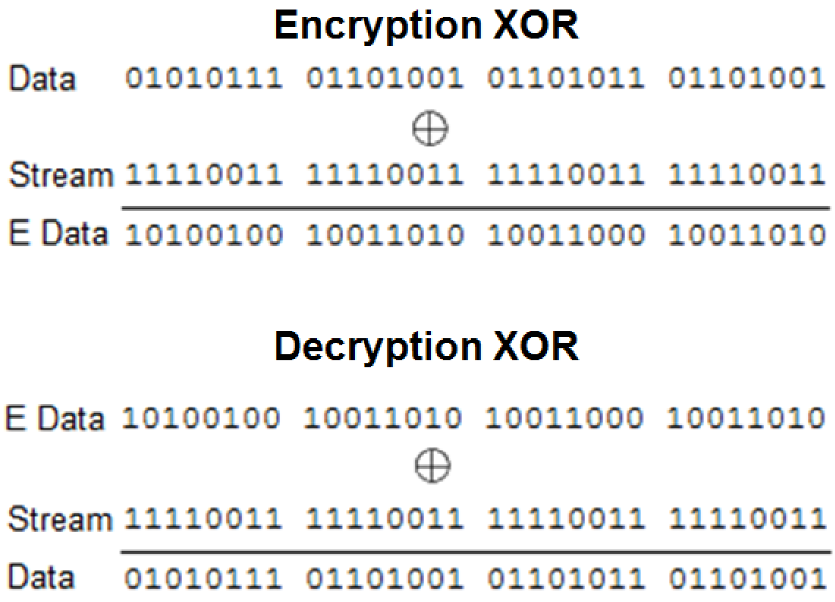


Figure 1: A image of rule 30 run for 15 iterations. The highlighted center column is chaotic and will be used for our stream cipher. Image courtesy of [7].

3. Run the CA m iterations where m is the number of bits in the data source. The bit of the center column of the CA is saved for use in the stream cipher.
4. Take the resulting center column bits and xor with data.

To decrypt the ciphertext, the stream is generated using the same method and XORed with the encrypted data the result is the original plaintext. Figure 2 shows how the XOR operator can be used to encrypt and decrypt data.



Encryption XOR

Data	01010111 01101001 01101011 01101001
	⊕
Stream	11110011 11110011 11110011 11110011
E Data	10100100 10011010 10011000 10011010

Decryption XOR

E Data	10100100 10011010 10011000 10011010
	⊕
Stream	11110011 11110011 11110011 11110011
Data	01010111 01101001 01101011 01101001

Figure 2: Sample encryption and decryption of data using a stream composed of the repeated binary sequence 11110011

3.4 Results

Several tests can be run to evaluate the effectiveness of the proposed encryption system. These tests include the entropy, % of bits that are 1, Matlab randomness test, key diffusion rate, and plaintext diffusion rate [4]. Each of these focuses on detecting the weakness of the encryption system against different types of attacks.

	Original Image	Encrypted Image
Entropy (10 is perfectly random)	7.717	9.813
% of bits that are 1	52.3%	50.5%
Matlab runs test	Fail	Pass
1 bit flip key diffusion rate	NA	49.92%
NPCR (1 bit flip plaintext)	NA	0.00003%

Table 1: Results of Rule 30 1 Dimensional Stream Cipher Encryption. The final two tests are only valid for an encryption system since they measure how effective the system is key and input text diffusion. Therefore there are no results for the original image.

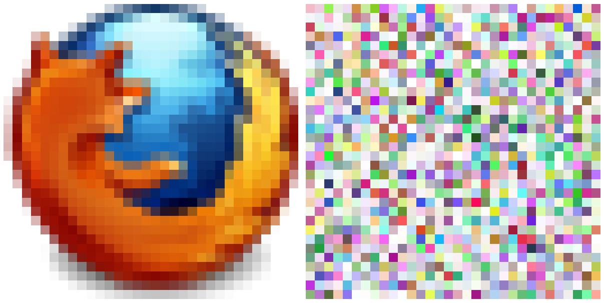


Figure 3: Original 32x32 png on the left. Encrypted png on the right.

3.4.1 Brute Force Key Search

The most basic form of attack is a brute force key search. In this type of attack an attacker will systematically try to decrypt the ciphertext by checking permutations of the key until the correct key is found.

The security of this encryption system against such an attack is based on the length of the key. For the tests run in this section a key of length 256 bits was used. To brute force a 256 bit key would require an attacker to check 2^{256} different keys. This key space is too large for an exhaustive key search using even the most powerful modern day supercomputers.

3.4.2 Cipher Only Attacks

In cryptography, a ciphertext-only attack (COA) or known ciphertext attack is an attack model for cryptanalysis where the attacker is assumed to have access only to a set of ciphertexts. Over the years cryptographers have developed statistical techniques for attacking ciphertext, such as frequency analysis. These attacks attempt to identify an underlying pattern in the data and exploit that pattern in order to reduce the number of attempts a brute force attack must make before decrypting the data. In order to protect against these attacks the cipher text should appear to be a random distribution of bits. This ability to generate an apparently random ciphertext is a property of the encryption system referred to as confusion.

One indication of random data is a uniform distribution of values. This can be checked by viewing a histogram of the distribution of red, green, blue, and alpha values of the pixels in the encrypted vs. original image. From figures 10 and 11 it is clearly shown that the distribution of rgba values is relatively uniform and differs from the original image.

A random dataset of 1s and 0s should have an approximately equal number of each type of bit in the set. A larger value of either indicates that the PRNG favors one bit over the other. This behavior decreases the encryption systems resistance to frequency attacks. The value for our encryption system is with one half a percent of 50%, indicating a pseudorandom distribution.

Another indication of randomness is image entropy. Image entropy is a statistical measure of randomness that can be used to characterize the texture of the input image. Entropy is defined as

$$E = -\sum_i (p. * log_2(p_i))$$

where p contains the histogram counts. A higher value of E is indicative of higher randomness in the image. From the results in table 1 we can see that entropy of the encrypted image is far greater than that of the plaintext image.

A more advanced test for randomness is the runs test used by Matlab. This test performs a runs test on the sequence of observations in the vector x. This is a test of the null hypothesis that the values in x come in random order, against the alternative that they do not. The test is based on the number of runs of consecutive values above or below the mean of x. Too few runs indicate a tendency for high and low values to cluster. Too many runs indicate a tendency for high and low values to alternate. The test returns the logical value h = 1 if it rejects the null hypothesis at the 5% significance level, and h = 0 if it cannot. From table 1 it is shown that the encrypted image passes the runs test while the original image fails the test.

3.4.3 Differential Attacks

Differential cryptanalysis is a general form of cryptanalysis applicable primarily to block ciphers, but also to stream ciphers and cryptographic hash functions. In the broadest sense, it is the study of how differences in an input can affect the resultant difference at the output. In the case of a block cipher, it refers to a set of techniques for tracing differences through the network of transformations, discovering where the cipher exhibits non-random behavior, and exploiting such properties to recover the secret key.

One indication of resilience to differential attacks is if an encryption system to has good diffusion. Diffusion means that a change a bit in the plaintext leads to several changes in the encrypted image. Ideally one would have a diffusion rate of approximately 50%, meaning that any change in the original plaintext appears to produce a completely different ciphertext.

When encrypting images the NPCR test can be used to test diffusion. NPCR rate is the % of bits that change when a single input bit is changed.

$$NPCR = \frac{\sum_i F(i)}{N} \times 100$$

where F(i) is the the bit value, 0 or 1, at the i index and N is the number of bits in the data set. From table 1 it is clear that our cipher has an extremely poor NPCR rate. This is due to the fact that the encryption systems cipher stream is generated solely based on the key with no regard the plain text. Thus, if one bit is flipped in the plain text, only one bit will be flipped in the ciphertext.

3.4.4 Key dependence analysis

Further attacks look at how the encryption system changes based on a change in the key. A good encryption system should produce a different output for different keys even if only one bit is flipped in the key.

To measure the key dependence of the encryption system a test was run where only one bit was flipped in the key and the number of flipped bits in the output was summed. From table 1 it is clear that even a small change in the initial key produces a completely different output.

3.4.5 Overview of Results

From table 1 and figure 3 the proposed encryption method was effective and producing a cipher image that differs from the original image and appears to be random noise.

Despite the success of this encryption method, there still exist some major drawbacks. The computation time of this encryption system is relatively large. This is due to the face that only one column of each CA iteration is used. This restriction forces the system to run at least one iteration for every bit in the plaintext. This is extremely inefficient and could never be used for any large dataset.

4 2D Life Like CA Stream Cipher

In this section the development of an encryption system using a chaotic life like CA will be discussed.

4.1 Choosing Life Like Chaotic CA

Several papers have looked at how effective various life like cellular automata behave as PRNGs. In [3] an encryption system was designed uses life like cellular automata. [3] studied several life like CAs for use as a PRNG and found that a Fredkin (B4357/02468) and Amoeba (B357, S1358) performed the best in a Diehard and Ent randomness tests. In the next section we will examine both CAs for use in our stream cipher.

4.1.1 Testing Chaos of CA

As a test to ensure that the Fredkin and Amoeba CA acts as a PRNG and is chaotic, the Lyapunov Exponent of the CA can be estimated in addition to running randomness tests on the output of the CA. A positive Lyapunov exponent is indicative of chaos.

The nondirectional maximum Lyapunov exponent (MLE) of an elementary cellular automaton (CA) may be interpreted as the natural logarithm of the time averaged number of cells c_j in a cell's neighborhood $N(c_i)$ that is affected during each consecutive time step if the state of c_j is perturbed. Results of testing Fredkin (B4357/02468) and Amoeba (B357, S1358) against the known chaotic rule 30 can be found in table 2.[1]

	Matlab Runs Test	Lyapunov Exponent	Lyapunov Exponent [3]
rule 30	Pass	0.653	NA
Fredkin (B4357/02468)	Pass	0.999	0.999
Amoeba (B357, S1358)	Pass	0.821	0.8604

Table 2: Comparison of calculated MLE values for different chaotic CAs. For this paper the MLE value was determined from the average of 5 runs and the MLE calculation between the 200^{th} and 201^{st} iteration with different initial conditions

The Fredkin CA outperformed the Amoeba CA in the tests from table 2 and the Diehard randomness test from [3]. Consequently, for our stream cipher we will use the Fredkin CA.

4.2 Proposed Algorithm

To encrypt a data set the following algorithm can be used.

1. Set the key as the initial condition of the CA by looping the key over a grid the size of the data set.
2. Run the CA for a number of iterations n.
3. Take the resulting array of bits and xor with data.

To decrypt the ciphertext, the stream is generated using the same method and XORed with the encrypted data, the result is the original plaintext.

4.3 Results

Testing of the effectiveness of a CA was discussed in section 3.4. In this section we will only summarize the results of using the life like CA.

	Original Image	Encrypted Image
Entropy (10 is perfectly random)	6.343	9.989
% of bits that are 1	47.05%	50.15%
Matlab runs test	Fail	Pass
1 bit flip key diffusion rate	NA	50.04%
NPCR (1 bit flip plaintext)	NA	0.000002%

Table 3: Results of Rule 30 1 Dimensional Stream Cipher Encryption. The final two tests are only valid for an encryption system since they measure how effective the system is key and input text diffusion. Therefore there are no results for the original image.

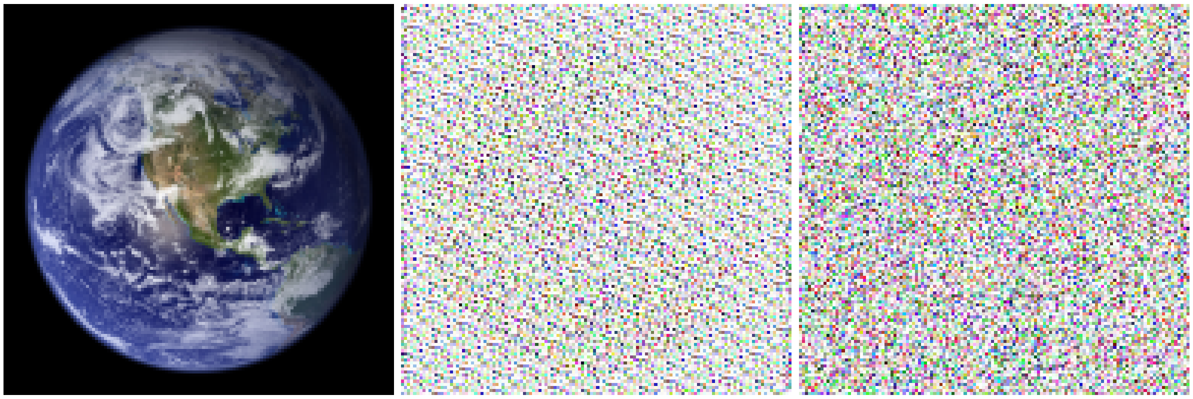


Figure 4: Original image on the left. Encrypted image 10 iterations in the center. Encrypted image 1000 iterations on the right.

The results displayed in table 3 are comparable to those generated by the 1 dimensional rule 30 stream cipher. However, the 2D cellular automaton was able to overcome many of the problems presented by the rule 30 stream ciphers. Most notably this algorithm allows all the bits of the final cellular automata to be used. Additionally, less iteration are required to generate a usable data stream.

From testing 100 to 1000 iterations appears sufficient to generate a pseudorandom stream. However, there is a danger that insufficient iterations will be used and remnants of the initial data will still be visible. Figure 4 shows an image after only 10 iterations. The outline of the earth is still visible indicating that there was not enough randomness in the CA after only 10 runs. As a rule of thumb the PRNG should be run a minimum of 100 iterations and until 99% of the rgba frequency values are within 10% to 15% of the mean frequency.

There still remains a major drawback in the design of this algorithm. The encryption systems NPCR remains low remains low leaving the ciphertext vulnerable to differential attacks.

5 Block Cipher

In order to produce an encryption system that has diffusion, a block cipher can be used. In cryptography, a block cipher is a deterministic algorithm operating on fixed-length groups of bits, called blocks, with an unvarying transformation that is specified by a symmetric key.

The modern design of block ciphers is based on the concept of an iterated product cipher. In cryptography, a product cipher combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components to make it resistant to cryptanalysis. Iterated product ciphers carry out encryption in multiple rounds, each which uses a different subkey derived from the original key.

One of the main advantages of block ciphers is diffusion. In our two previous encryptions systems, the encrypted data was still vulnerable to a differential attack. Due to the use of S and P boxes, to be explained later, block ciphers are able to have a high sensitivity to the input text.

5.1 Block Cipher Algorithm

The proposed block cipher algorithm can be broken into several parts.

1. Generation of keys.
2. Generation of P and S Boxes
3. Block Cipher Encryption Algorithm
4. Block Cipher Chaining

An overview of the block cipher encryption system without chaining can be viewed in figure 5. This particular block cipher is a substitution permutation network. Each part of the system servers a particular focus.

S-Box In cryptography, an S-Box (Substitution-box) is a basic component of symmetric key algorithms which performs substitution. They are typically used to obscure the relationship between the key and the ciphertext.

P-Box In cryptography, a permutation box (or P-box) is a method of bit-shuffling used to permute or transpose bits across S-boxes inputs, retaining diffusion while transposing.

Sub Key In a block cipher, a rounding function is used to derive several keys from the original key that can be XORed with the input to each round of S-Boxes. In figure 5 $K_{0,1,2,3}$ are the result of the rounding function. These keys will be referred to as sub keys.

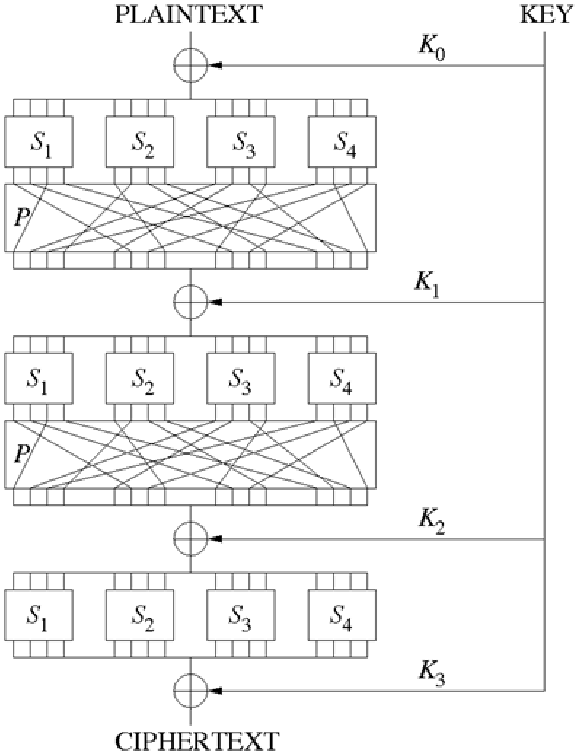


Figure 5: A sketch of a Substitution-Permutation Network with 3 rounds, encrypting a plaintext block of 16 bits into a ciphertext block of 16 bits. The S-boxes are the S_{is}, the P-boxes are the same P, and the round keys are the K_{is}.

5.1.1 Generation of keys

In order generate the sub keys used in the substitution-permutation network the Fredkin (B4357/02468) CA can be used. A n x m grid can be initialized where n is the size of the key and m is number of keys that are needed. For this algorithm the number of keys needed is 4. The n x m grid is initially filled with the repeating value of the key. The system is then run for several iterations to produce a number of psudorandom keys. The small grid size allows a random pattern to appear in less iteration. After only 10 iterations the pattern from the original key was destroyed by the chaotic nature of the CA. To make the computation easier a 32 x 32 array is used to represent the 256 x 4 array of keys. The output after only 10 iterations can be viewed in figure 6.

5.1.2 S-Boxes

S-boxes are a key part of an effective block cipher. One of the simplest forms of a S-box is where a n bit number is input into the s box and is transformed into another n bit number via

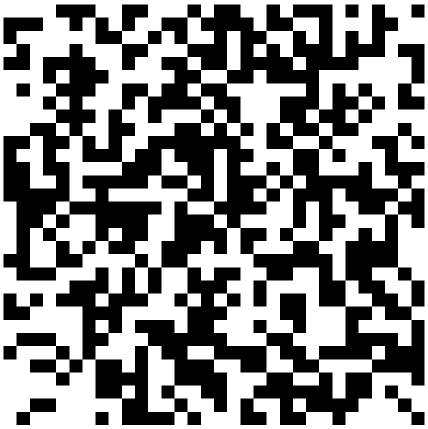


Figure 6: An array of subkeys generated by the Fredkin CA after 10 iterations

a lookup. An example of a 16 bit s box can be seen in table 4. S-boxes where multiple inputs map to the same output are forbidden since such a box would make the system non reversible. 32 32 bit S-boxes were used in this block cipher encryption system.

input	output
0	6
1	15
2	2
3	12
4	11
5	1
6	9
7	8
8	13
9	7
10	4
11	14
12	3
13	10
14	5
15	0

Table 4: A S-Box input output map

5.1.3 P-Box

P-boxes ensure that our block cipher has good diffusion. An example of how a p box functions can be seen in figure 5. For the p boxes used in our algorithm we generated 2 random P-boxes ensuring that at least one bit from each S-box was mapped to one of the 8 S-boxes in the following row.

5.1.4 Mode of Operation

In cryptography, mode of operation is the procedure of enabling the repeated and secure use of a block cipher under a single key, also referred to as chaining. A block cipher by itself allows encryption only of a single data block of the cipher's block length. When targeting a variable-length message, the data must first be partitioned into separate cipher blocks. Typically, the last block must also be extended to match the cipher's block length using a suitable padding scheme. A mode of operation describes the process, type of chaining, of encrypting each of these blocks, and generally uses randomization based on an additional input value, often called an initialization vector, to allow doing so safely.

One of the advantages of chaining is to avoid the same input production the same ciphertext. Figure 7 shows the result of a block cipher encryption system without chaining.

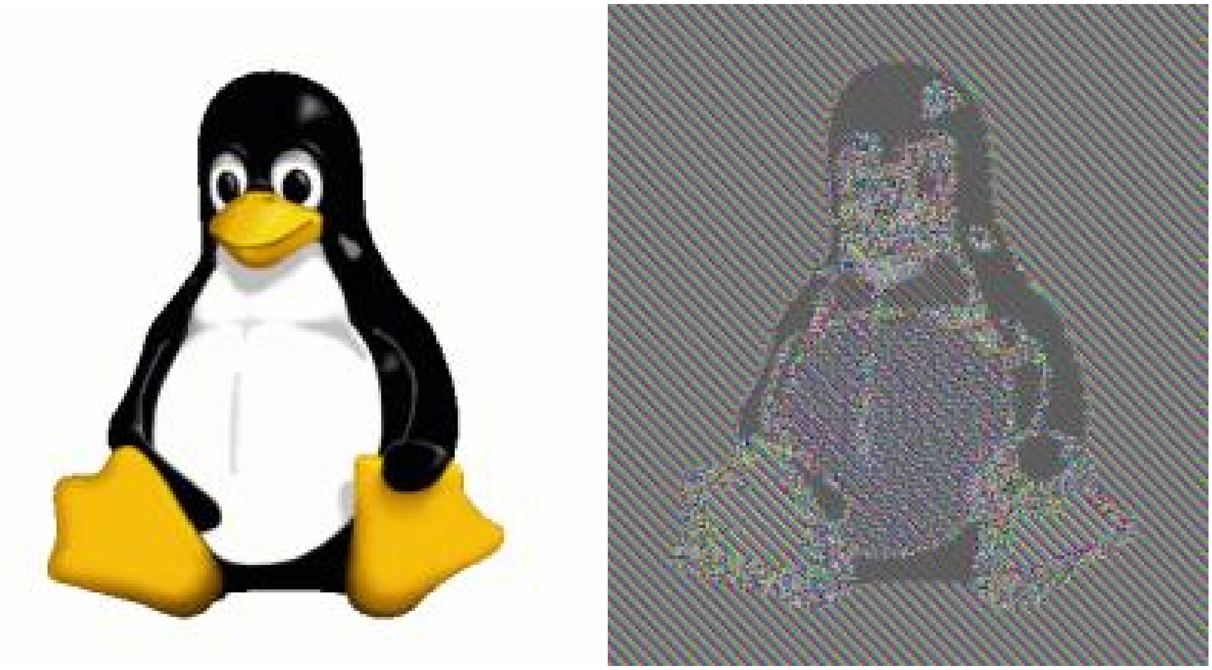


Figure 7: The image on the left shows the original image. The image on the right is encrypted using a block cipher without chaining.

For this block cipher algorithm a Propagating Cipher Block Chaining (PCBC) mode of encryption will be used. The encryption and decryption process is described in figure 8.

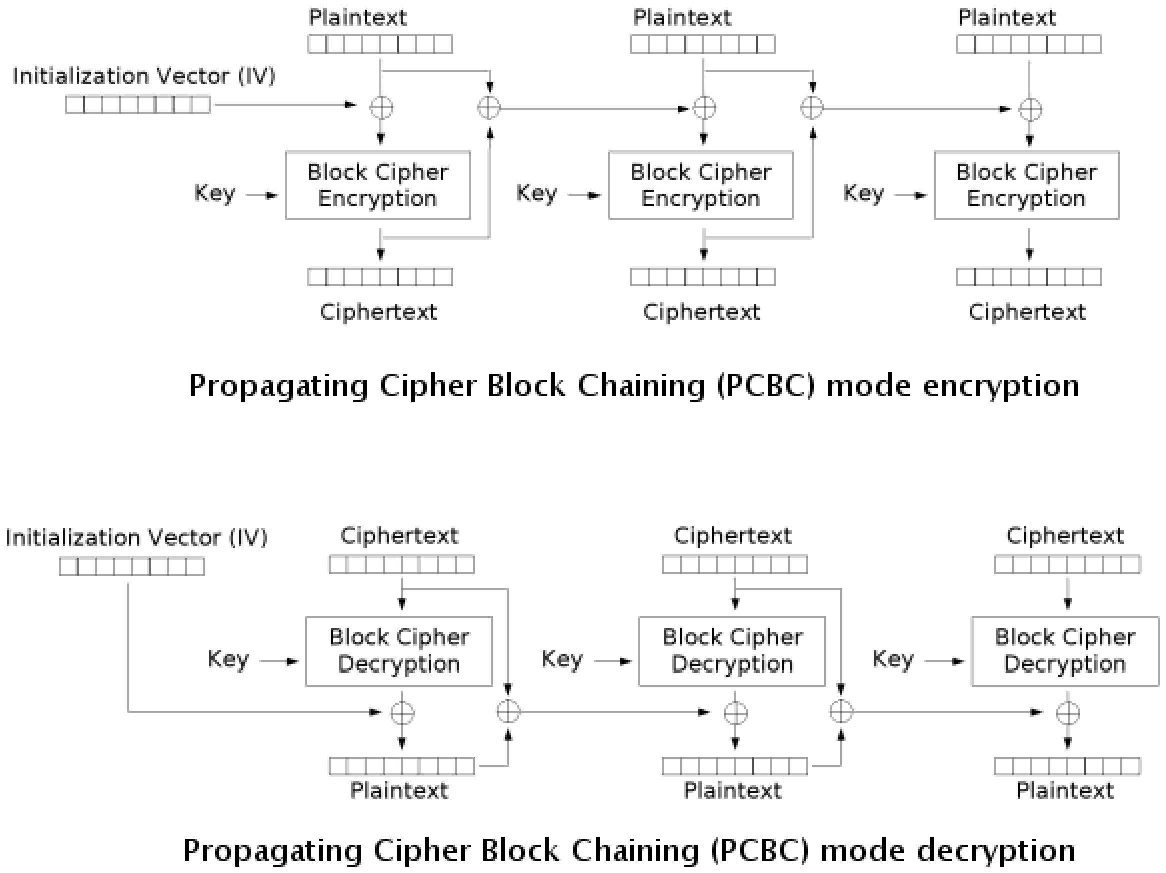


Figure 8: A sketch of the propagating cipher block chaining mode of operation used in this block cipher encryption system. The initialization vector is a randomly generated binary vector of length 256 bits.

5.2 Results

Testing of the effectiveness of a CA was discussed in section 3.4. In this section we will only summarize the results of using the Frekin CA as basis for the block cipher described in the previous subsections.

The results displayed in table 5 and figures 9, 14, and 15 are comparable to those generated by the first two encryption methods. However, the major difference in the results is the increase in the NPCR value. This result indicates that the block cipher method is more resilient to differential attacks then the stream cipher methods.

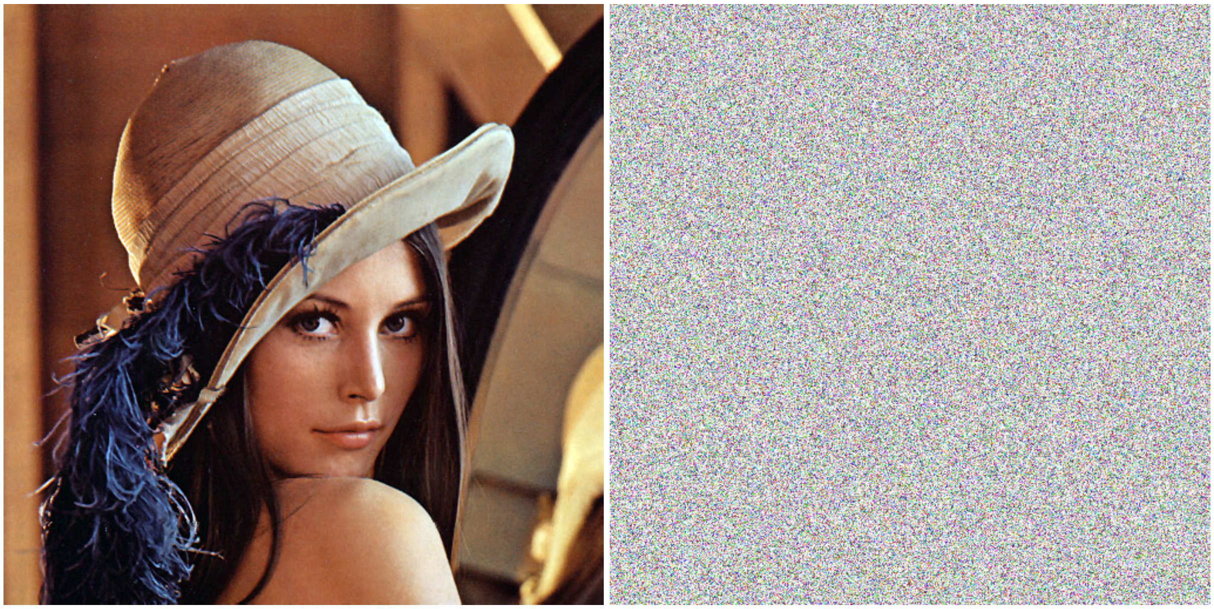


Figure 9: Results of encryption of 512x512 png using the block cipher method described in this section. The original image is on the left. The encrypted image is on the right. Just in case you couldnt tell.

	Original Image	Encrypted Image
Entropy (10 is perfectly random)	7.325	9.9993
% of bits that are 1	58.68%	49.98%
Matlab runs test	Fail	Pass
1 bit flip key diffusion rate	NA	49.93%
NPCR (1 bit flip plaintext)	NA	48.01%

Table 5: Results of block cipher encryption based on the Fredkin CA. The final two tests are only valid for an encryption system since they measure how effective the system is key and input text diffusion. Therefore there are no results for the original image.

6 Conclusion

To conclude, we will expand on the joint results of all three encryptions systems, closely examine their strengths and weaknesses, and propose extensions.

6.1 Overall Results

One of the most valuable results is that all three chaotic cellular automata encryption systems had excellent confusion. All of the systems exhibited high entropy, high key diffusion, and good bit frequency as well as passing the Matlab runs test. Despite this common characteristic the stream ciphers failed to exhibit diffusion while the block cipher demonstrated excellent diffusion due to the S and P boxes.

The encryptions systems were further separated by their computational requirements. The 1D rule 30 stream cipher proved unrealistic due to its exorbitant memory and computational costs when dealing a data stream of larger than a few thousand bits. The 2D Fredkin stream cipher fared better by limiting the memory usage to the size of the data stream being encrypted. This limited data size kept the computational requirements for each iteration constant leading to a significant improvement over the 1D rule 30 encryption system. However, the best performance was the block cipher. Combining multiple permutations together with a small size CA grid composed of the sub keys significantly reduced the computational requirements.

While the block cipher was able to outperform the stream ciphers under normal conditions, it lacks the parallelism that is available in the stream ciphers. Table 6 shows how calculating CAs using a Nvidia CUDA program can reduce computation time by several fold. It would also be possible to further increase performance with a hardware implementation such as the one suggested in [6].

CA	Grid Size	Iterations	Python	C	CUDA
Rule 30	128 to 8320 bits	4096	323s	0.205s	0.046s
2D Fredkin	512 x 512 bits	1000	954s	2.156s	0.049s

Table 6: Results of running rule 30 and Fredkin CA using python, C, and CUDA. Pythons slow performance was due to the use of lists instead of python arrays. Additionally, 0.035s of the CUDA code was spent sending data to and from the GPU. For larger iterations or grid sizes CUDA would increase its lead over c and python

6.2 Strengths, Weaknesses, and Extensions

As mentioned above all of the encryptions systems performed extremely well when basic cryptanalysis tests were performed. However, to ensure that the output of the encryption system produces an excellent pseudorandom ciphertext more advanced randomness tests could be run. In [3] the diehard and ent tests were used to test the randomness of the life like CAs. These randomness tests could be used on the output of our encryption systems to ensure that the output is pseudorandom.

One weakness with using CAs for cryptography is the danger of weak keys. In cryptography, a weak key is a key, which, used with a specific cipher, makes the cipher behave in some undesirable way. Weak keys usually represent a very small fraction of the overall keyspace, which usually means that, if one generates a random key to encrypt a message, weak keys are very unlikely to give rise to a security problem. Nevertheless, it is considered desirable for a cipher to have no weak keys. A cipher with no weak keys is said to have a flat, or linear, key space. Testing for weak keys is a difficult task, however, there is one week key for all of the encryption systems in this paper. A key of all 0s would produce a grid of all zeros in the stream ciphers and all of the sub keys in the block cipher would be comprised of only zeros. To avoid this problem a key of all zeros could be forbidden for use with these encryption systems.

One weakness of the block cipher encryption system is the possibility of weak S boxes. For example the DES encryption system was originally designed with weakens S boxes. However, the NSA mysteriously changed the S boxes when the government was reviewing the DES system. The 8 S-boxes of DES were the subject of intense study for many years out of a concern that a backdoor might have been planted in the cipher. The S-box design criteria were eventually published [2] after the public rediscovery of differential cryptanalysis, showing that they had been carefully tuned to increase resistance against this specific attack. Other research had already indicated that even small modifications to S-boxes could significantly weaken encryption systems.

7 Appendix

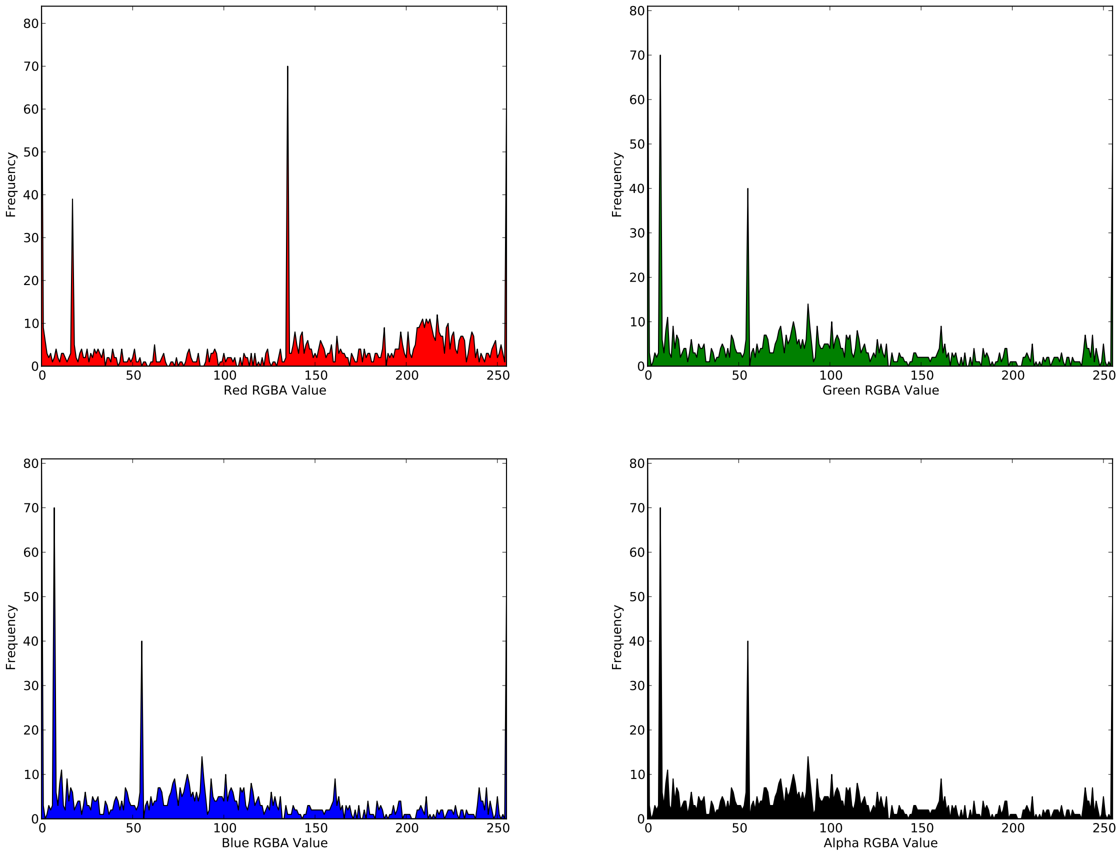


Figure 10: This histogram shows the distribution of rgba values in the original test image in figure 3

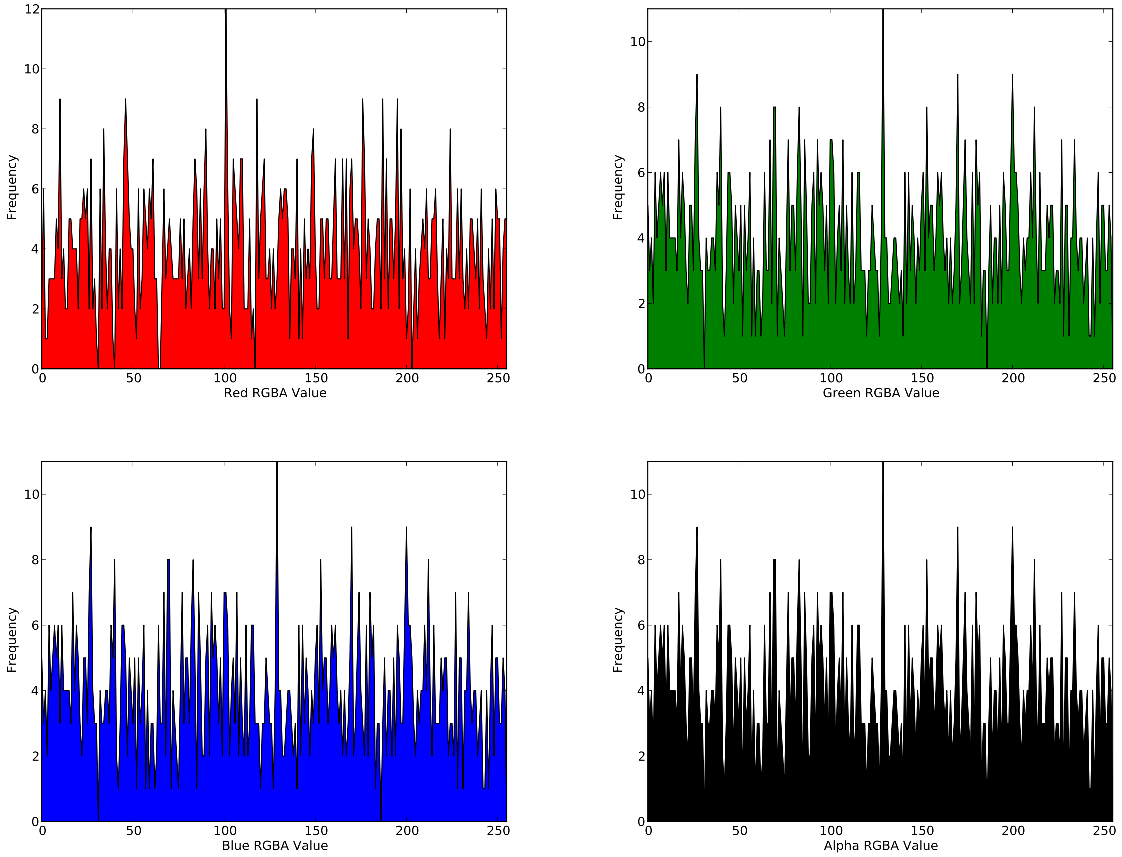


Figure 11: This histogram shows the distribution of rgba values in the encryption image in figure 3

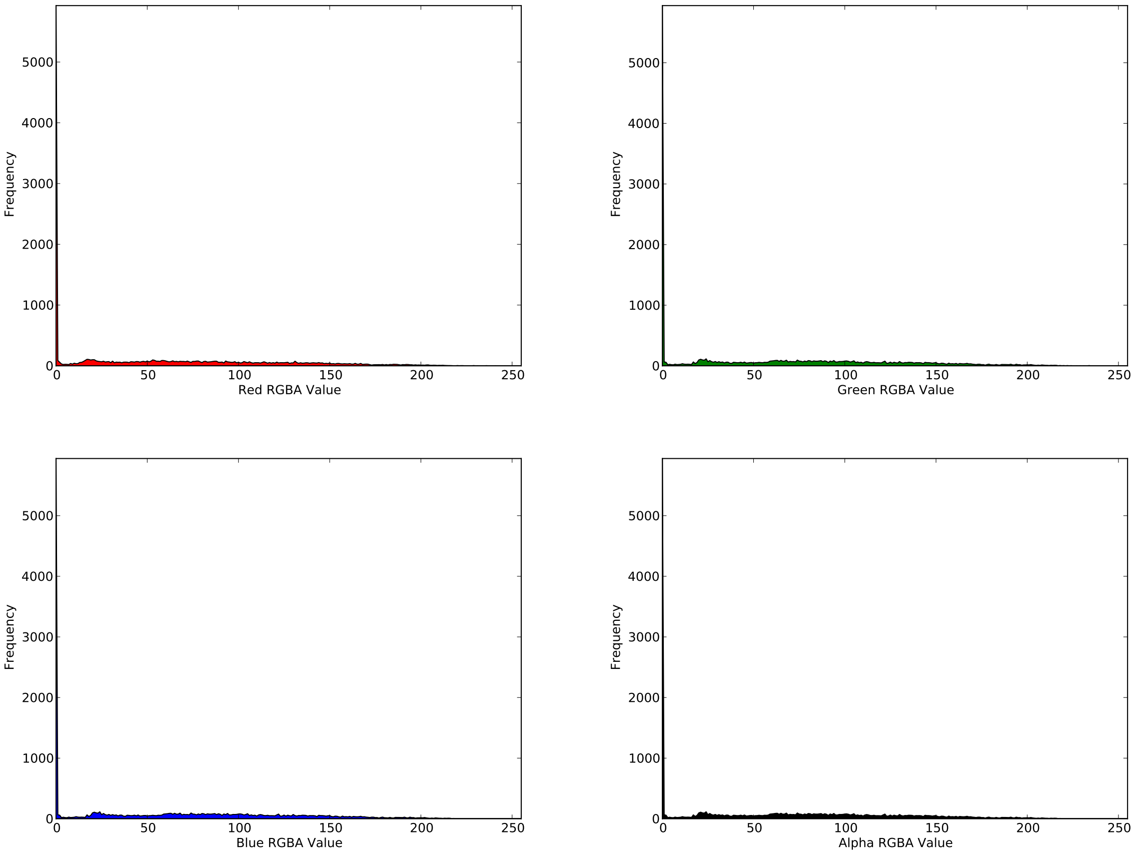


Figure 12: This histogram shows the distribution of rgba values in the original test image in figure 4

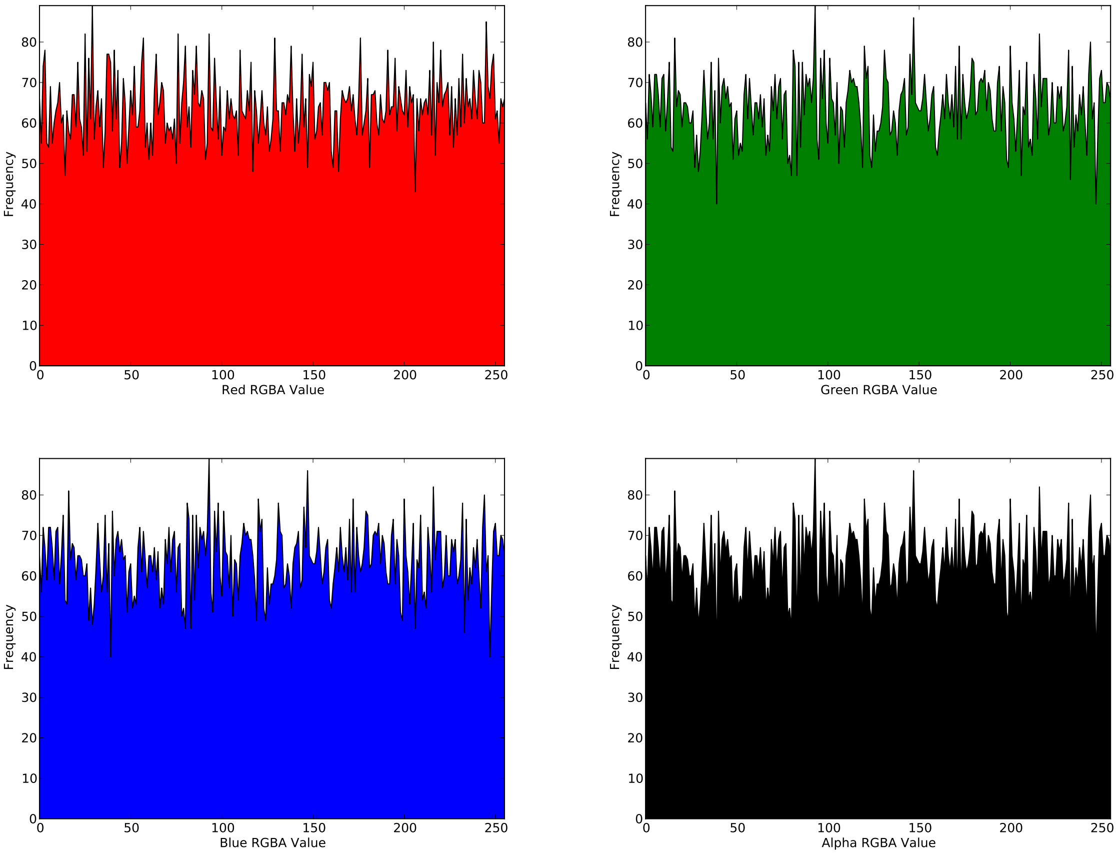


Figure 13: This histogram shows the distribution of rgba values in the original test image in figure 4

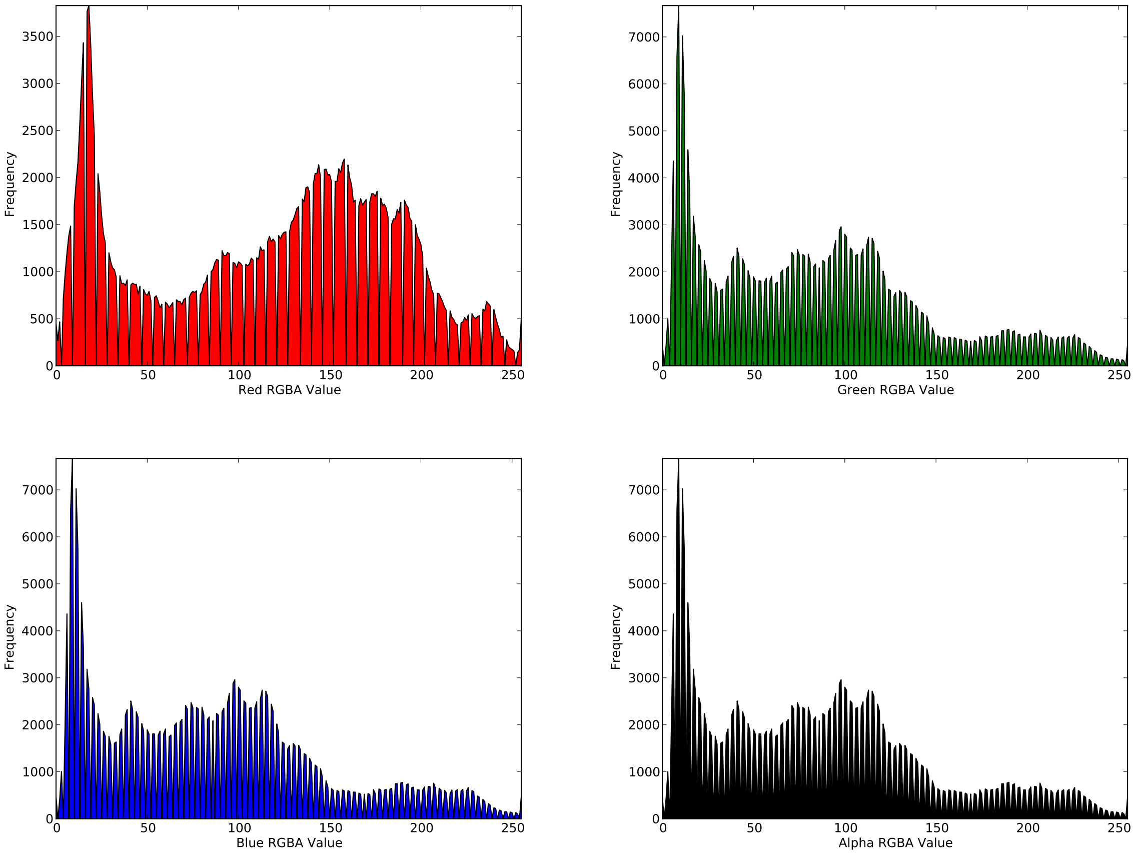


Figure 14: This histogram shows the distribution of rgba values in the original test image in figure 9

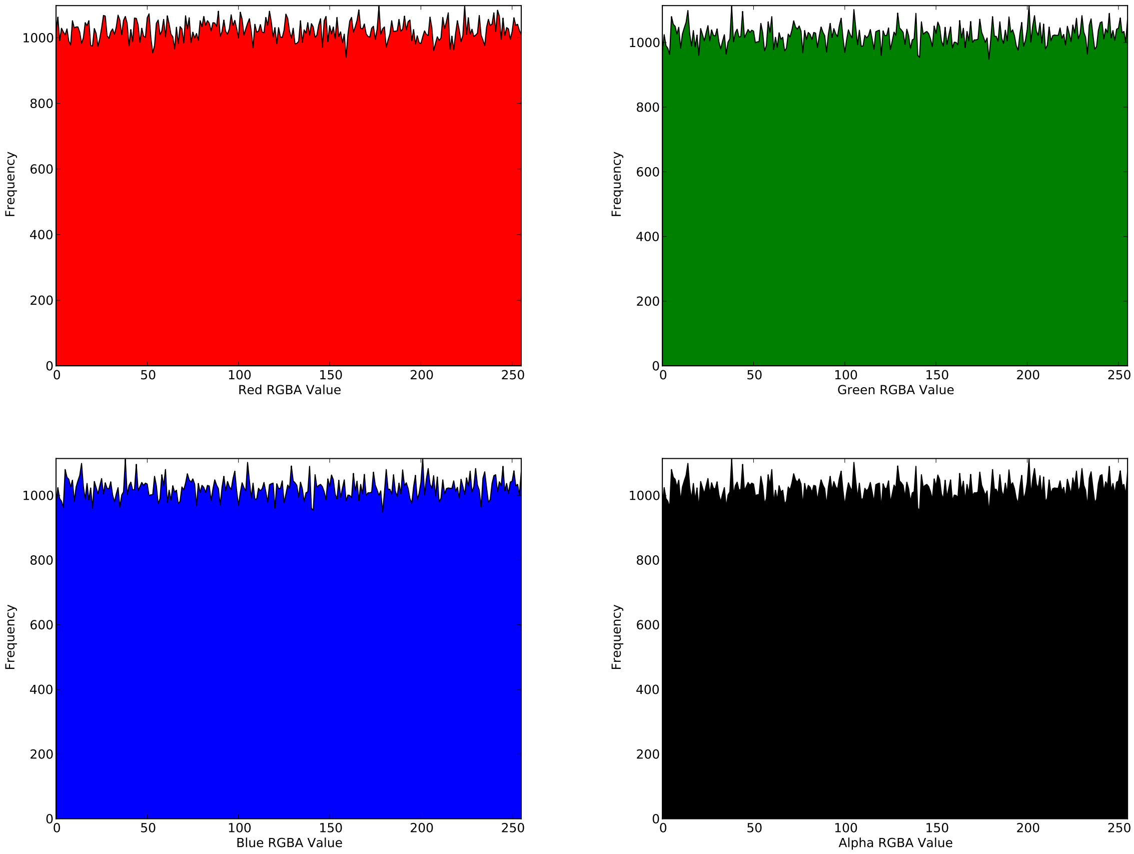


Figure 15: This histogram shows the distribution of rgba values in the original test image in figure 9

References

[1] Jan Baetens. Lyapunov exponents of elementary cellular automata. `http://demonstrations.wolfram.com/`, 2010.

[2] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *Instituto de Fisica De Sao Carlos*, 38:243–250, 1994.

[3] Odemir M. Bruno Marina Jeanth Machicao, Anders G. Marco. Chaotic encryption method based on life-like cellular automata. *Instituto de Fisica De Sao Carlos*, 2011.

[4] Saeed Rahati-Q Zohreh souzanchi-k Maryam Habibipour, Mehdi Yaghobi. An image encryption system by indefinite cellular automata and chaos. *2010 2nd International Conference on Signal Processing Systems*, 2010.

[5] S.C. Phatak. Logistic map as random number generator. `http://www.iopb.res.in/~phatak/numsim/node14.html`, 2004.

[6] Somanatha Tripathy and Sukumar Nandi. Lcase: Lightweight cellular automata-based symmetric-key encryption. *International Journal of Network Security*, 8:243–252, 2009.

[7] Eric W Weisstein. Rule 30. `http://mathworld.wolfram.com/Rule30.html`, 2003.

[8] Stephen Wolfram. *A New Kind Of Science*. Wolfram Media, Champaign, IL, 2002.

Chaotic Exploration
Applications to Reinforcement Learning

Rowan Wing
Department of Computer Science
University of Colorado, Boulder
rowan.wing@colorado.edu
May 5th, 2012

Abstract

I propose integrating the use of chaotic trajectories in reinforcement learning tasks to regulate agent exploration. Reinforcement learning [11] has shown itself to be a powerful optimization tool with broad applications across disciplines. Reinforcement learning, similar to much of human and animal learning, relies on the process of trial and error to shape the agent's interaction with its environment. The method used to mitigate the tension between exploitation and exploration during these trial and error interactions has significant impact on the performance of the reinforcement learning algorithm. In most reinforcement learning algorithms, the process of exploration ultimately relies on a pseudo-random number generator. By integrating chaotic mappings into the exploration policy in place of pseudo-random number generators, we see significant improvement in performance on a series of simple maze tasks when compared to similar algorithms using pseudo-random number generators.

1 Introduction

Reinforcement learning theory originated in the field of psychology as a model for human learning. In the last couple of decades it has been embraced by the machine learning community as an effective method for optimization and a theoretical tool for understanding learning in artificial agents. Similar to a human learner, the virtual agent learns thorough experience with its environment. As the agent navigates its environment, it is given feedback in the form of numerical rewards, which it uses to form judgments about the value of its actions. Because the agent's learning is experiential, it must mediate the conflict between exploitation and exploration. To artfully maintain the balance between these two conflicting impulses, the learning agent requires a structured and directive policy, but one with enough flexibility to allow for novelty. This paper presents the use of chaotic trajectories to guide the agent's course of action between exploitation and exploration.

2 Reinforcement Learning

2.1 Context

Machine learning is grossly divided into supervised and unsupervised learning. In supervised learning, the agent's learning is directed by training data, which has been labeled by a human expert. Feedback provided to the agent in supervised learning is direct and informative. The labeled training data provides feedback not only on the correctness of the choice the agent made, but instructs the agent about the proper choice if the agent chooses in error. A classic example of supervised learning is a classifier - for example, an email spam filter.

In unsupervised learning, in contrast, the agent must learn without the benefit of a objective truth metric. Most unsupervised learning is concerned with inferring structure in unlabeled data sets. A classic example of unsupervised learning is a clustering algorithm, for example, a recommender system or topic modeler. Although reinforcement learning is sometimes placed under unsupervised learning algorithms, it is more commonly given its own class. Similar to unsupervised learning algorithms, reinforcement learning distinguishes itself from supervised learning by operating without labeled training data or directive feedback. However, it also differs from other unsupervised learning algorithms because of its focus on interactions with an environment.

Reinforcement Learning algorithms learn through experience with an environment using the process of trial and error. The agent is not told which actions to pursue, but instead must search through the environment and discover which action choices yield the greatest cumulative return. This process of search and trial and error decision making distinguish reinforcement learning from other forms of machine learning. Reinforcement learning models represent their environment with a set of states, $s \in \mathcal{S}$, and actions, $a \in \mathcal{A}$, available from those states. The goal of reinforcement learning is to establish a policy, denoted π_t, which maps perceived states to actions at time, t, to maximize a cumulative return function. The policy is developed probabilistically as the agent navigates the environment. When an agent takes an action, a, from state, s, it receives feedback about its choice in the form of a single numeric value, called the reward, r_t. This reward is used to adjust the policy values and the probability, $\pi_t(s,\ a)$, of taking action a from state s, in the future.

2.2 Popular Algorithms

There are three broad classes popularly applied to reinforcement learning [11]: dynamic programming, Monte Carlo methods, and temporal difference learning. Dynamic programming refers to a set of model-based algorithms used for optimization. Similar to divide and conquer algorithms, dynamic programming works by breaking a problem down into subproblems, solving these subproblems, and recombining their solutions [1]. In reinforcement learning applications, dynamic programming uses a perfect model of the environment as a Markov decision process and divides the updating process recursively. Although dynamic programming is an effective optimization technique, and useful for reinforcement learning theoretically, it is rarely implemented in practice because of the requirement of a perfect model for the environment and its computational cost.

Monte Carlo methods, in contrast to dynamic programming, can operate effectively in the absence of a complete model of the environment. Monte Carlo methods use the average of sampled data and therefore are effective with only partial experience of the possible environmental states. Monte Carlo methods can be applied to reinforcement learning either in an on-line or simulated context. Although a model is required for the simulated application, in which simulated experience is used for policy development, it requires only a partial model of the environment. Monte Carlo methods have the disadvantage that they are episodic in nature and are unable to bootstrap. Although their episodic nature gives Monte Carlo methods some protection against violations of the Markov property, it also leads to slower convergence times.

Temporal difference learning combines ideas from dynamic programming and Monte Carlo methods. Like dynamic programming, temporal difference learning operates in an on-line fashion. Temporal difference learning algorithms update value estimates incrementally without waiting for the final outcome, i.e. they bootstrap. Like Monte Carlo methods, they are experienced based and do not assume a complete model of the environment. Temporal difference learning algorithms epitomize the process of reinforcement learning [11].

2.3 Temporal Difference Learning

There are a number of temporal difference learning paradigms, but the most popular are the temporal difference control algorithms, Sarsa [9, 11] and Q-learning [7, 8, 11, 13]. I examined two versions of Sarsa and three versions of Q-learning. The first version of Sarsa and Q-learning I worked with were both one-step methods in which updates are made to the current state only, denoted Sarsa(0) and Q(0). In both cases, the value of the state action pair $Q(s, a)$, is updated based on a weighted temporal difference, or error (δ), between the current estimated value of $Q(s,\ a)$ and the sum of the reward received when taking action a plus the discounted value of the state-action pair resulting from taking action a, $Q_t(s',\ a')$. The difference between Sarsa(0), which is an on-policy control algorithm, and Q(0), which is an off-policy control algorithm, is in how the value of the next state-action pair is computed. In Sarsa(0), the value of $Q_t(s',\ a')$ is determined from following the current exploration policy to generate a'. In Q(0) the value of $Q_t(s',\ a')$ is determined from following a greedy exploration policy at state s' to generate a'. In Q(0) the action a' will always be the max-value over all possible choices, denoted a^*. The two algorithms are shown below.

Sarsa(0):

$$
\begin{aligned}
&Q_{t+1}(s,\,a) = Q_t(s,\,a) + \alpha\delta \\
&\delta = r + \gamma Q_t(s',\,a') - Q_t(s,\,a)
\end{aligned}
\tag{1}
$$

Q(0):

$$
\begin{aligned}
&Q_{t+1}(s,\,a) = Q_t(s,\,a) + \alpha\delta \\
&\delta = r + \gamma Q_t(s',\,a^*) - Q_t(s,\,a), \\
&a^* = max_{a'}\, Q_t(s',\,a')
\end{aligned}
\tag{2}
$$

In both cases the learning rate (α), $0 < \alpha \leq 1$, is the weight given to new information. When $\alpha = 0$ the agent does not learn from experience. As $\alpha \to 1$, the weight given to new experience increases. The other parameter, the discount rate (γ), $0 \leq \gamma \leq 1$, indicates the importance of future rewards. When $\gamma = 0$, the update of $Q_t(s,\,a)$ ignores the value of the next state, s'. As $\gamma \to 1$, the importance of the value of $Q_t(s',\,a')$ increases in the update of the current state-action pair $Q_t(s,\,a)$.

The other three reinforcement learning implementations incorporated eligibility traces. An eligibility trace is a method for assigning credit to previous states when a reward is received. In the one-step methods described above, credit is only assigned to the state immediately preceding the rewarded state. In a two-step method, credit for the reward would be passed back two states and similarly with n-step methods. A popular n-step eligibility trace, called the λ-return [11], is defined by (3) below in which $\mathcal{R}_t^{(n)}$ is the n-step return at time t.

$$
\mathcal{R}_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\mathcal{R}_t^{(n)},\ 0 \leq \lambda \leq 1 \tag{3}
$$

The λ-return is a particular way of averaging n-step backups in which the weight given to a preceding state fades by λ at each step. The $(1-\lambda)$ coefficient is a normalizing factor that ensures the weights sum to one. When $\lambda = 0$, the λ-return reduces to the one-step backup, e.g. Sarsa(0) and Q(0), and when $\lambda = 1$, the λ-return is the same as the full Monte Carlo algorithm. Figure 1 below gives a pictorial representation of eligibility traces. In this example, a non-zero reward is received at the cell marked with a star (*) and the size of the arrow in the second and third frame denote the amount of credit assigned to each preceding cell. We can see from the third frame that the use of λ-returns can greatly increase the speed of learning since the effect of the reward echoes back through the path taken.

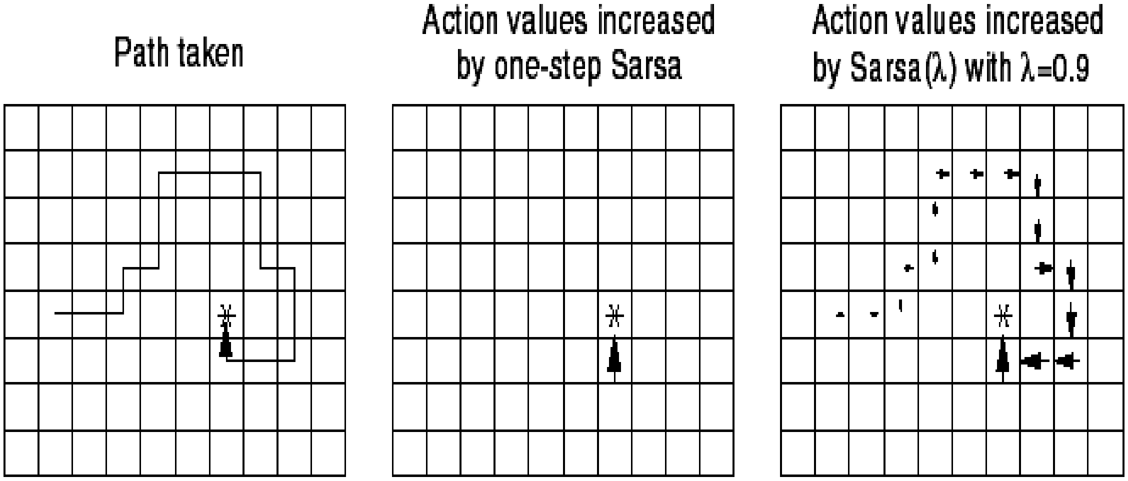

Path taken
Action values increased by one-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9

Figure 1: Shows the benefit to learning that eligibility traces can offer. Image taken from [11].

The three following algorithms, Sarsa(λ), Watkin's Q(λ), and Peng's Q(λ), are similar to the Sarsa(0) and Q(0) algorithms presented in (1) and (2), but they also incorporate λ-return backups. The difference between Watkin's Q(λ) and Peng's Q(λ) is subtle. In Watkin's Q(λ), eligibility traces are only updated if the action taken, a', equals the greedy action, a^*. Otherwise the eligibility trace is set to zero. In Peng's Q(λ), eligibility traces are updated on exploratory as well as greedy actions. The concern with only updating eligibility traces for greedy actions is that cutting off traces every time an exploratory step is taken, loses much of the advantage of using eligibility traces. Peng's implementation of Q(λ) aims to address this concern and maintain learning speed in the face of exploration.

Sarsa(λ):

$$\begin{aligned} Q_{t+1}(s,\, a) &= Q_t(s,\, a) + \alpha\delta e(s,\, a) \\ \delta &= r + \gamma Q_t(s',\, a') - Q_t(s,\, a) \\ e_t(s) &= \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases} \end{aligned} \tag{4}$$

Watkin's Q(λ):

$$
\begin{aligned}
&Q_{t+1}(s,\, a) = Q_t(s,\, a) + \alpha\delta e(s,\, a) \\
&\delta = r + \gamma Q_t(s',\, a^*) - Q_t(s,\, a) \\
&a^* = max_{a'}\, Q_t(s',\, a') \\
&e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \text{ and } a' = a^* \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{5}
$$

Peng's Q(λ):

$$
\begin{aligned}
&Q_{t+1}(s,\, a) = Q_t(s,\, a) + \alpha\delta' e(s,\, a), \text{ for all } Q_t(s,a) \\
&Q_{t+1}(s,\, a) = Q_t(s,\, a) + \alpha\delta, \text{ for current } Q_t(s,a) \\
&\delta = r + \gamma Q_t(s',\, a^*) - Q_t(s,\, a) \\
&\delta' = r + \gamma Q_t(s',\, a^*) - Q_t(s,\, max_a\, Q_t(s,\, a)) \\
&a^* = max_{a'}\, Q_t(s',\, a') \\
&e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}
\end{aligned} \tag{6}
$$

Here again the learning rate (α), $0 < \alpha \leq 1$, is the weight given to new information. The discount rate (γ), $0 \leq \gamma \leq 1$, indicates the importance of future rewards. The two new parameters, $e_t(s)$ and λ, both relate to the incorporation of the eligibility trace. The eligibility trace, $e_t(s)$, is the distribution of credit for a reward and λ, $0 \leq \lambda \leq 1$, is the weight given to the trace.

2.4 Exploration

The five temporal difference equations presented above dictate how the value functions are updated when an action is taken, but it is the policy itself that determines which action is taken. The action selection algorithm in many ways is the most important part of reinforcement learning. It is the process that determines to which states in the environment the agent has access and is ultimately responsible for all the information the agent receives. Although there are numerous exploration algorithms, I used two standard ones for comparison, ϵ-greedy and softmax.

There is a natural tension in reinforcement learning between exploitation and exploration. Exploitive policies are greedy and choose the best action at any given time, where "best" is defined as the action with the highest perceived reward. Policies which allow for exploration, in contrast, will allow for a sub-optimal action in the short term in hope of achieving a greater ultimate return in the long run. Policies at one extreme or the other do not perform well. Policies that are too exploitive run the risk of getting stuck in local optima, whereas policies that are too exploratory continually deviate from the chosen path, even when they find a global optimum, and may fail to converge.

An ϵ-greedy exploration policy mediates the conflict between exploitation and exploration by choosing a greedy or exploratory step with a preset probability. The parameter ϵ controls the probability for exploration as shown in (7) below. For most of this project I set $\epsilon = 0.05$, a value used in [11] and one which showed good results empirically when compared with other values for ϵ.

$$\pi_t(s,\, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{for exploitive steps} \\ \epsilon & \text{for exploratory steps} \end{cases} \tag{7}$$

A softmax exploration policy chooses actions with probabilities based on the relative values of $Q_t(s,\, a)$. I used the Boltzmann or Gibbs distribution as shown below. The probability of each action a, $\pi_t(s,\, a)$, is weighted by a temperature parameter, τ, and then normalized over all possible action choices from the current state, s. As $\tau \to 0$ the policy becomes increasingly greedy and as $\tau \to \infty$ the policy becomes increasingly random. The value for τ is often static, although it can be adjusted dynamically as I demonstrate in section (3).

$$\pi_t(s,\, a) = \frac{e^{Q_t(s,\, a)/\tau}}{\sum\limits_b e^{Q_t(s,\, b)/\tau}} \tag{8}$$

2.5 Comparison of Reinforcement Learning Algorithms

To establish a baseline, I ran the five algorithms above on two simple map environments. Map 1 is the more pathological of the two, in which the optimal path is buried under four low-value choices. All agents start at Node 1 and the probability of randomly finding the optimal path is already only 0.016. However, every time the agent encounters the -2 reward on its way to the optimal path, it becomes increasingly less likely to choose that path again. Map 2 is a little more friendly and the optimal path is easier for agents to find. Agents in Map 2 start in Node A and it only takes one sub-optimal step to put the agent on the optimal path.

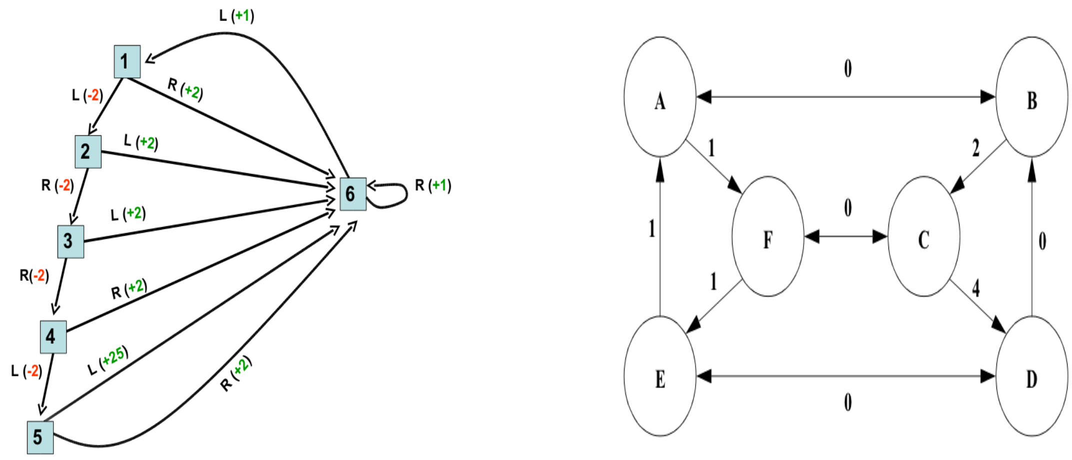


Figure 2: The left frame shows Map 1 and the right frame shows Map 2

In the experiments I ran each agent through 1,000 trials, called an episode. To evaluate the progress of the agent's learning, I examined the agent's $Q_t(s,\, a)$ values after each trial. The $Q_t(s,\, a)$ values are a measure of how much the agent has learned about the environment at any point in time. To evaluate the fitness of the agent's learning, I used a greedy policy and ran it on the $Q_t(s,\, a)$ values for an entire episode (1,000 trials) and calculated the expected return. By running this process, I had a measure of learning at each trial number that I could plot. However, before I plotted the results, I ran 100 episodes and averaged the returns to help eliminate noisy runs. The parameters α, γ, and λ were optimized using a simple grid search on the interval [0, 1] separately for each algorithm to ensure optimal performance. The values used are given in parentheses after the algorithm's name in each figure.

Figure 3 shows the algorithms' performance on Map 1 using ϵ-greedy exploration, and (Fig. 4) shows the algorithms' performance on Map 2 using ϵ-greedy exploration. In both (Fig. 3) and (Fig. 4), the horizontal red dashed line represents the optimal reward for the environment. The lines are colored according to the algorithm. We can see that on Map 1 all the algorithms get stuck in a sub-optimal policy, with two of the Q-learning algorithms performing only slightly better. In Map 2 we can see that all algorithms perform better than on Map 1 and again the Q-learning algorithms outperform the Sarsa algorithms. For the remainder of the project I focus only on Q-learning algorithms since they perform better than the Sarsa algorithms in these environments.

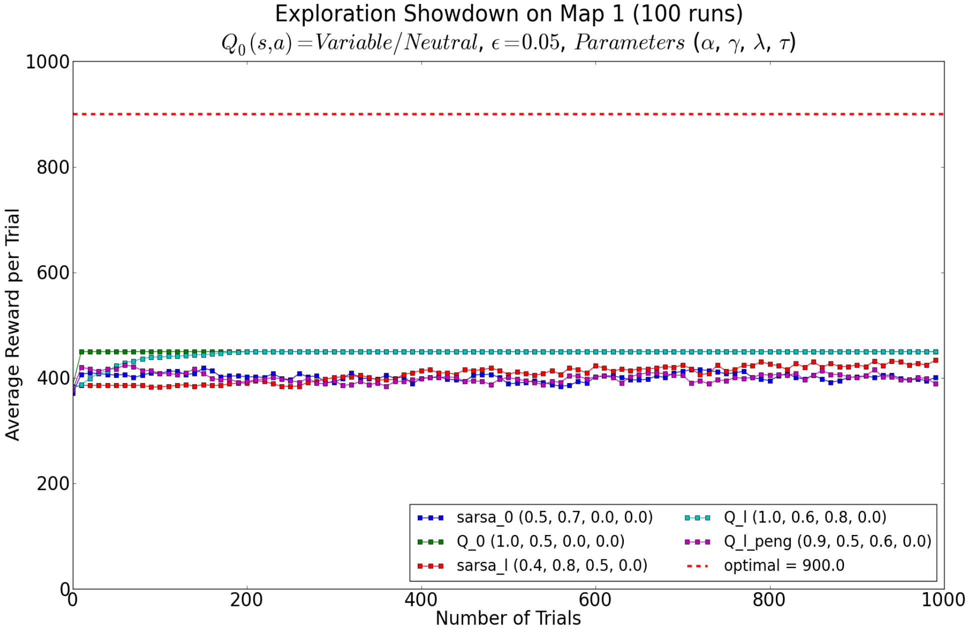


Figure 3: Performance comparison on Map 1

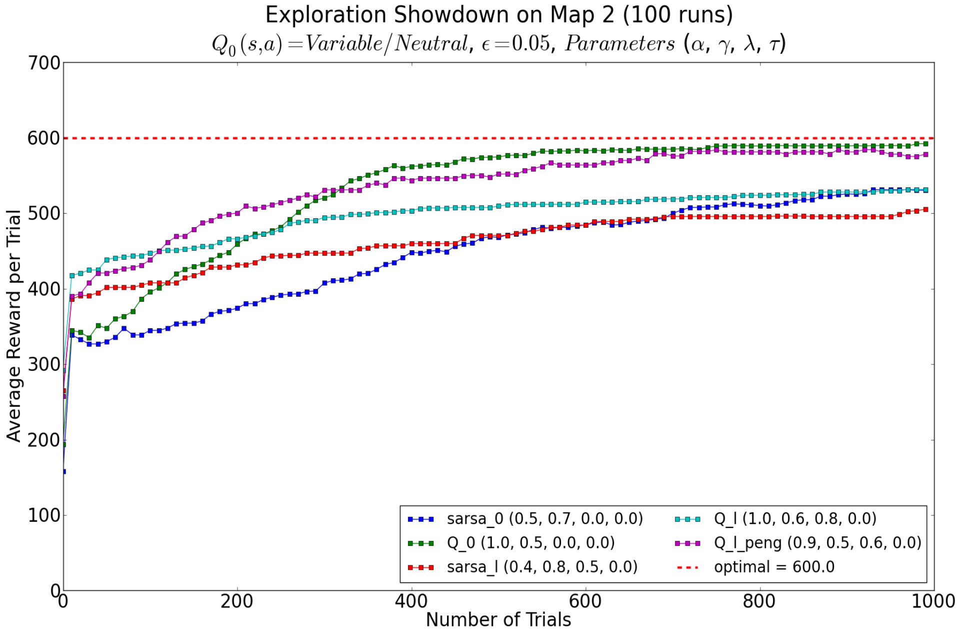


Figure 4: Performance comparison on Map 2

3 Chaotic Exploration

Traditionally reinforcement policies, such as ϵ-greedy and softmax, relay on uniformly random pseudo-random number generators when deciding whether to take an exploratory or exploitative step. However, similar "random" behavior has been shown to result from the use of deterministic chaotic trajectories [6]. This section outlines some ways in which chaotic systems can be exploited to increase the performance of traditional reinforcement learning algorithms.

3.1 Logistic Map vs ϵ-greedy

Recently, the idea of applying chaotic systems to reinforcement learning has gained attention. The logistic map, $x_{t+1} = 4.0x_t(1 - x_t)$, has been used in combination with an ϵ-greedy policy by [4,5]. I replicated their results using the same coefficient value and starting with $x_0 = 0.2$. The results are shown in (Fig. 5). The algorithms using a built-in pseudo-random number generator are shown with a square marker and the algorithms using the logistic map as their number generator are shown with a circular marker. In the Map 1 environment, there was a slight increase in the performance of Q(λ), although it was not significant. On Map 2 we see a significant increase in the speed of convergence from all three algorithms using the logistic map as a number generator. For the remainder of the project I focus only on Map 1 since it appears to be the harder of the two environments to learn.

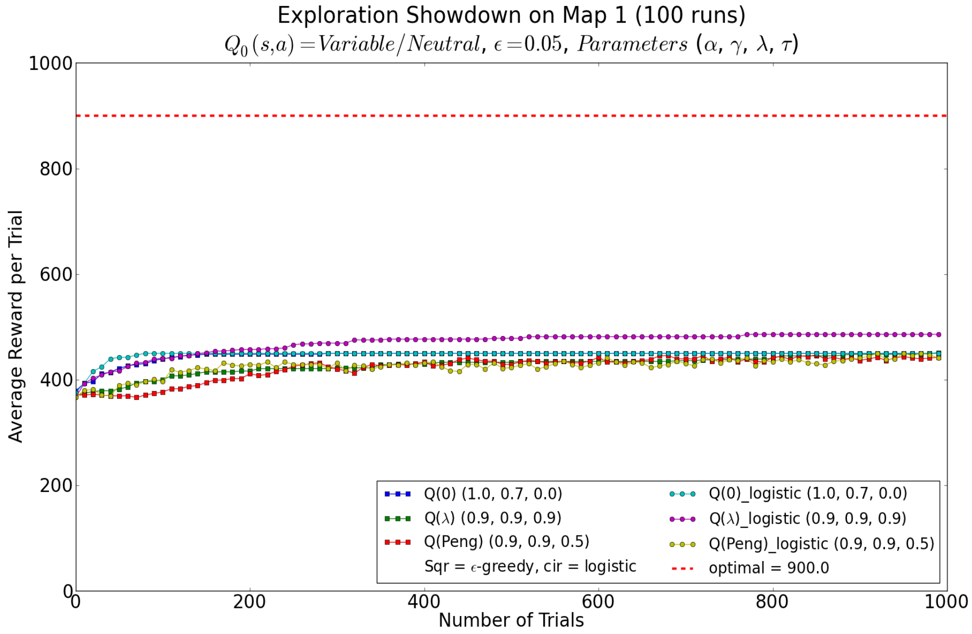


Figure 5: Performance comparison on Map 1 between a pseudo-random number generator and the logistic map

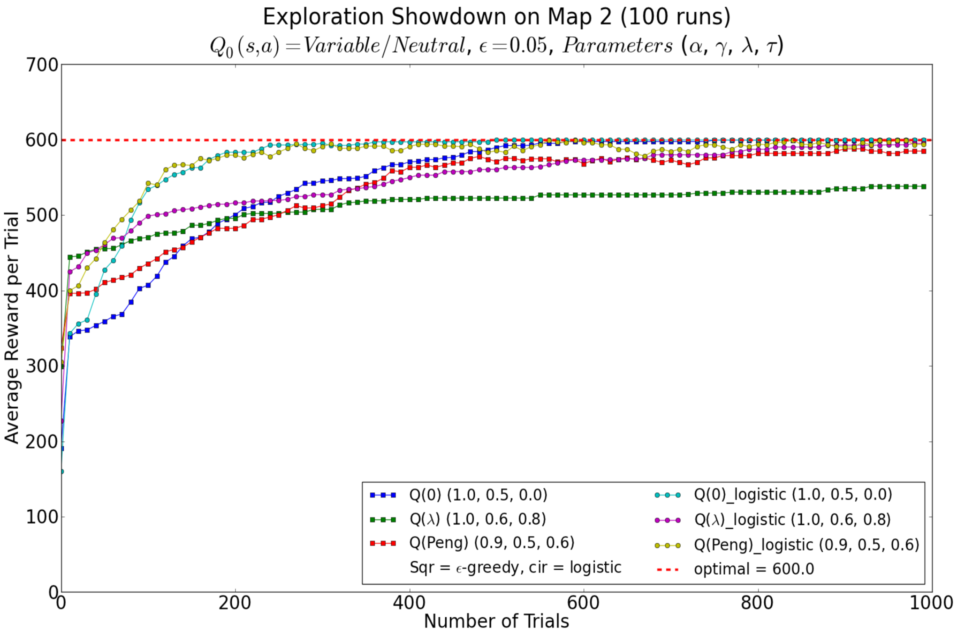


Figure 6: Performance comparison on Map 2 between a pseudo-random number generator and the logistic map

Both map environments favor exploration since agents need to pass through at least one sub-optimal choice from where they are started to find the optimal path. With closer examination, we see that this is exactly what the logistic map based number generator provides to the agent. Whereas the built-in Python pseudo-random number generator provides an essentially uniform distribution, the logistic map is heavily weighted near zero and one. Figure 7 shows the two distributions over one million iterations. I believe it is the increased opportunity to explore which allows the algorithms based on the logistic map to perform well in environments that encourage exploration or in stochastic environments [4, 5].

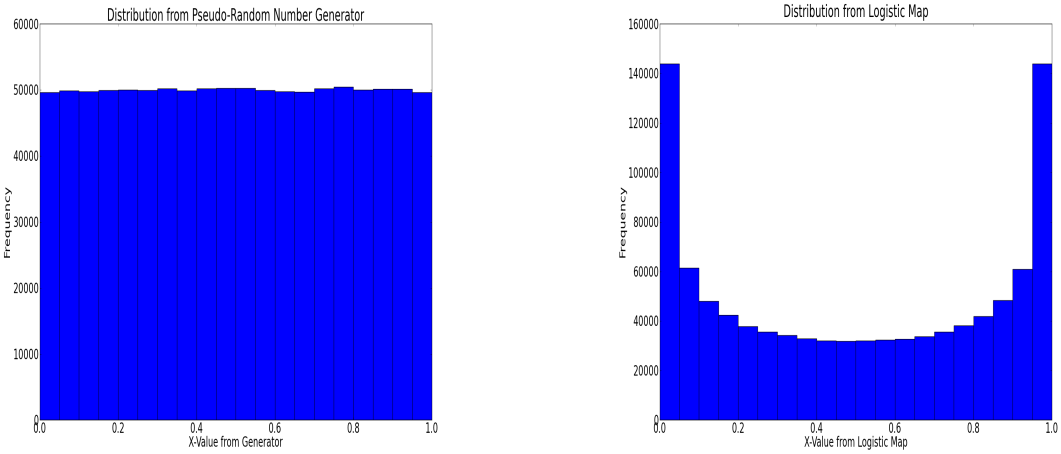


Figure 7: The left frame shows the distribution of numbers from Python's built-in pseudo-random number generator and the right frame shows the distribution from the logistic map, $x_{t+1} = 4.0x_t(1 - x_t)$, $x_0 = 0.2$.

3.2 Chaos in Softmax

Another alternative to the ϵ-greedy exploration policy, is the softmax policy described in (8). The probabilities given by the softmax policy are determined by the value of $Q_t(s,\, a)$ and the value of τ, the temperature parameter. The value of τ is often held constant throughout the learning process, but can also be annealed over an episode. Higher values of τ equalize the probabilities of the choice options and lead to more exploration. Lower values of τ lead to greedy action selection. Intuitively, we want an annealing schedule that provides higher values of τ initially to facilitate exploration and lower values for τ near the end of the episode to facilitate convergence. I used a mapping onto the Lorenz system, inspired by [2, 3], to generate the τ-value at each step.

Initially, following the values for the Lorenz system given by [2,3], I produced two trajectories with slightly different initial conditions using a Runge Kutta 4 integrator. I mapped the second trajectory onto the first and used the differences in the x-value to generate (Fig. 8) below. Unfortunately, because of the Lorenz system's sensitivity to initial conditions, the resulting plot was the inverse of the desired policy. It had small differences initially and increasingly large and unpredictable values as time progressed. Instead, I altered the Lorenz system parameters to create a two-cycle attractor[1] and a single-point attractor. A mapping, as described above, on these two parameterizations of the Lorenz system, yields the the plots in (Fig. 9) below. The rate of convergence can be controlled by changing the initial conditions and/or the system parameters.

[1] Thank you to Yogesh Virkar for suggesting the parameters to accomplish this in his class presentation

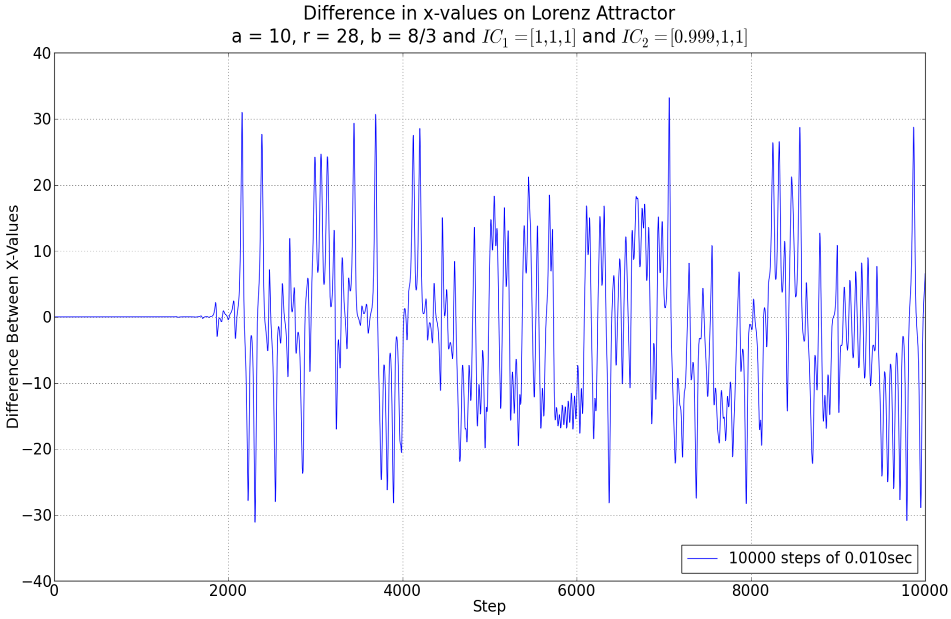


Figure 8: The difference in x-coordinates from the initial conditions (1,1,1) and (0.999,1,1) when mapped onto the Lorenz system defined by a=10, r=28, b=8/3

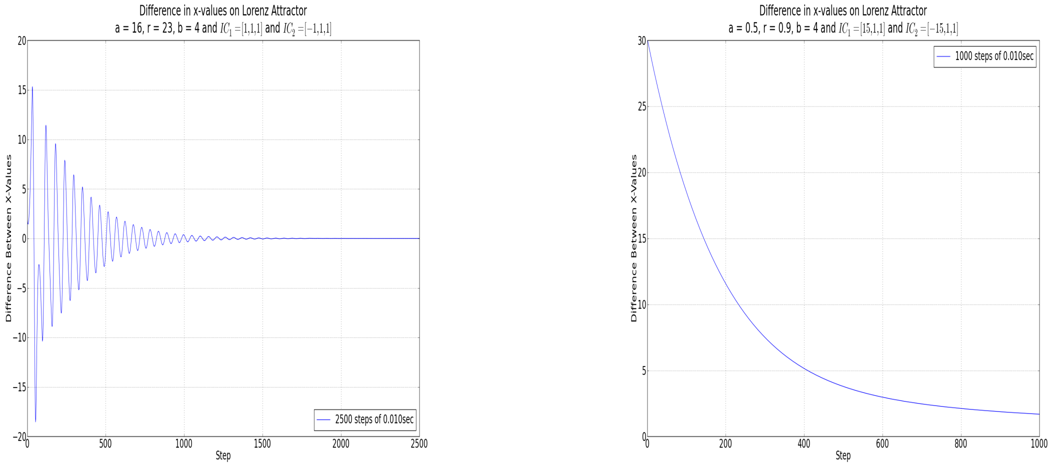


Figure 9: The left frame shows the x-differences of the initial conditions (1,1,1) and (-1,1,1) when mapped onto the Lorenz system defined by a=16, r=23, b=4. The right frame shows differences from the initial conditions (15,1,1) and (-15,1,1) mapped onto the Lorenz system defined by a=0.5, r=0.9, b=4

This method, which I denoted τ-mapping, provides significant improvement over the logistic mapping on Map 1. Figure 10 shows a comparison using the two-cycle attractor. All three algorithms are able to break away from the sub-optimal path and converge very near the optimal path. The speed of convergence of the $Q(0)$ and $Q(\lambda)$ is especially noteworthy. A mapping using the single-point attractor, (Fig. 11), likewise shows improvement over the logistic mapping, but falls short of the first τ-mapping.

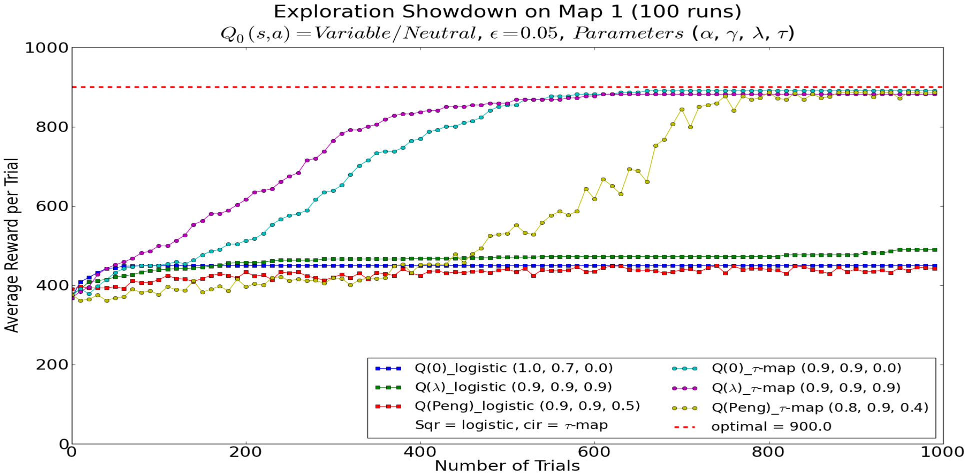


Figure 10: Perfomance comparison between the logistic mapping technique of [4, 5] and my τ-mapping method using the two-cycle Lorenz attractor

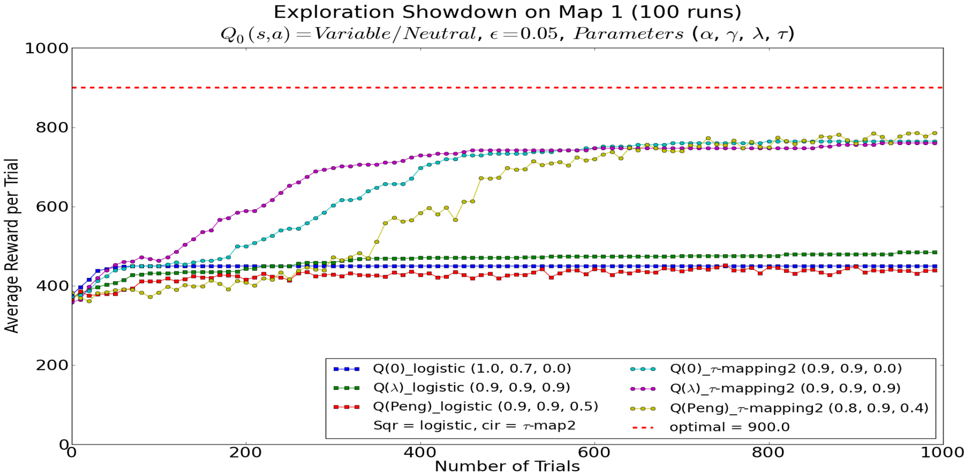


Figure 11: Perfomance comparison between the logistic mapping technique of [4, 5] and my τ-mapping method using the single-point Lorenz attractor

4 Discussion

Although initial results show promise for integrating chaotic systems into existing reinforcement paradigms, the more I push on my results, the less convinced I am. Algorithms based on the logistic map were shown to improve performance in stationary maps, for example on my Map 2 environment and in experiments by [4, 5]. This increase in performance was even more pronounced in non-stationary learning environments [4, 5]. However, by my analysis, it appears that this increase in performance is due simply to an increased predilection for exploration provided by the non-uniform distribution of the logistic mapping. Looking at the distributions in (Fig. 7), the logistic map is roughly three times more likely to explore than the standard ϵ-greedy algorithm based on the pseudo-random number generator. I increased the ϵ-value of the standard algorithm to $\epsilon = 0.15$ and compared its performance to the algorithm based on the logistic map with an ϵ-value of $\epsilon = 0.05$ to test my conjecture. The result is shown in (Fig. 12). We can see that by simply increasing the ϵ parameter we can match the performance of the chaotic-based exploration policy.

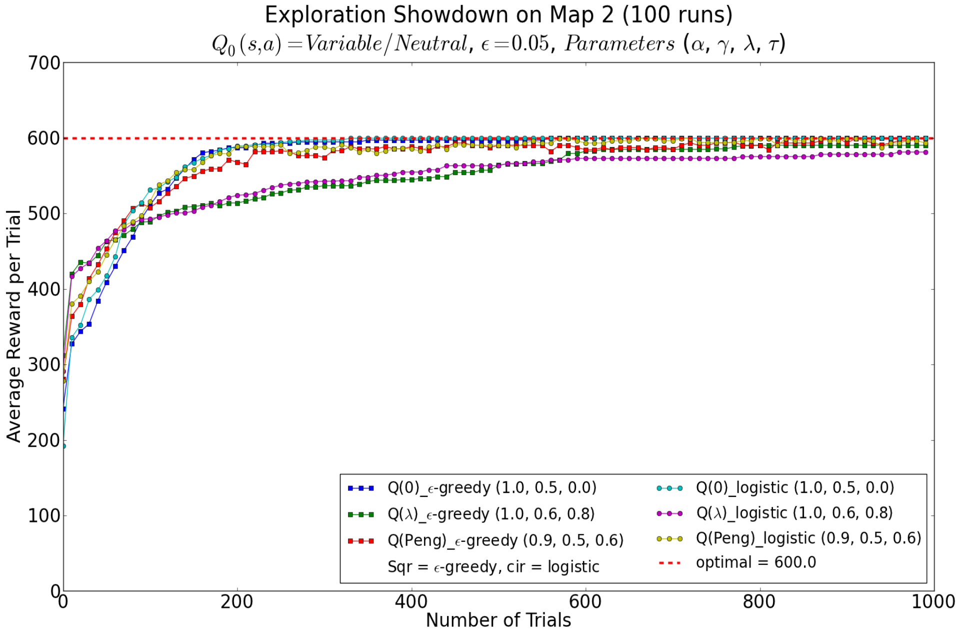


Figure 12: Perfomance comparison between the logistic mapping technique of [4, 5] and a standard ϵ-greedy algorithm with a higher ϵ-value, 0.15 compared with 0.05

Unfortunately, I saw the same pattern with my method using the τ-mapping. I compared my results, which had seemed quite significant when compared to the logistic map algorithms, with standard softmax exploration policies. I experimented with different initial values for τ and a constant and exponential decay model for the annealing schedule. I found that a very high initial value for τ paired with a constant annealing schedule showed performance on par with my τ-mapping algorithm. The results on the $Q(0)$ and $Q(\lambda)$ were almost identical, although the performance of my τ-mapping on Peng's $Q(\lambda)$ continued to show significant performance benefits over the standard algorithm. The results are shown below in (Fig. 13). Admittedly, I started τ at an "unnaturally" high value, viz. 1,000, but nonetheless the standard algorithm could match the performance of the chaotic mapping in two out of the three cases with a simpler implementation. Although these results do not discourage me from the belief that incorporating chaotic trajectories into the exploration poilicy is fruitful, the results make me cautious about tauting the benefit of chaotic-based methods without first aggressively pushing on the existing algorithms.

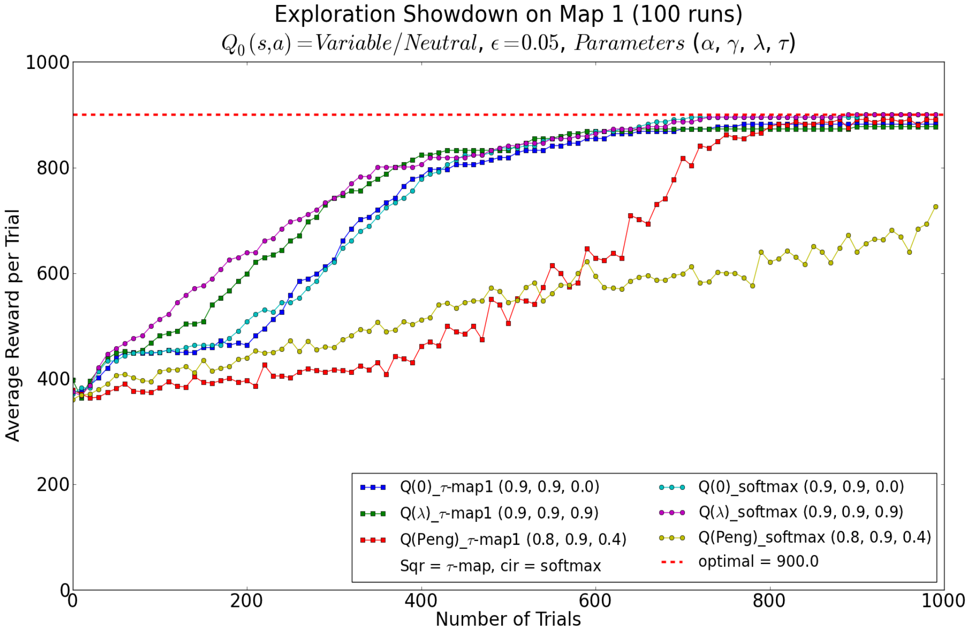


Figure 13: Perfomance comparison between my τ-mapping technique and a standard softmax algorithm with a very high initial τ-value, $\tau = 1,000$, and a constant annealing schedule

5 Conclusion

The distinction between randomness and chaos is compelling. Individual trajectories are unpredictable in both random and chaotic systems [6], yet chaotic attractors are magnificent in their structure when viewed from a macro-perspective. Learning appears to follow a similar trajectory. When viewed on an individual level, it is impossible to predict what a single person will learn or struggle with in the future, yet viewed more globally, there are distinct patterns of development for all typically-developing children. Human learning appears to benefit from an underlying structure and it is appealing to expect that artificial learning will similarly benefit from such a structure.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 2009.

[2] D. Dabby. *Musical Variations from a Chaotic Mapping.* PhD thesis, Massachusetts Institute of Technology, 1995.

[3] D. Dabby. Musical variations from a chaotic mapping. *Chaos*, 6, 1996.

[4] K. Morihiro, T. Isokawa, N. Matsui, and H. Nishimura. Effects of chaotic exploration on reinforcement learning in target capturing task. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 12, 2008.

[5] K. Morihiro, N. Matsui, and H. Nishimura. Chaotic exploration effects on reinforcement learning in shortcut maze task. *International Journal of Bifurcation and Chaos*, 16(10), 2006.

[6] T.S. Parker and L.O. Chua. *Practical Numerical Algorithms for Chaotic Systems.* Springer-Verlag, 1989.

[7] J. Peng and R. Williams. Incremental multi-step q-learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 226–232, 1994.

[8] J. Peng and R. Williams. Incremental multi-step q-learning. *Machine Learning*, 22:283–290, 1996.

[9] G.A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[10] S.H. Strogatz. *Nonlinear Dynamics and Chaos.* Westview Press, Cambridge, Massachusetts, 1994.

[11] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachusetts, 1998.

[12] H. van Hasselt. *Insights in Reinforcement Learning.* PhD thesis, University of Utrecht, 2010.

[13] C. Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge, England, 1989.

Sleepy Chaos

By Jeff Winship
May 5th, 2012

Introduction

Sleep is an interesting phenomenon that is just shrugged off by people as a totally normal process. However, if you think about it critically, a person is willfully going unconscious for several hours, hallucinating during the process, and potentially talking/moving while unconscious. This has always fascinated me, and because other areas of the human body, like heart rhythms, have been shown to have chaotic patterns, this natural process also seems ripe for analysis based on the fundamentals of chaos. In this paper we will be focusing specifically on trying to calculate a Lyapunov exponent for a person's movement frequencies while asleep.

Background

Some additional background on sleep patterns are required in order to understand exactly what is going on while someone sleeps. All humans (indeed, all land mammals) require at least some minimal amount of sleep to continue functioning, and this sleep serves various functions. One extremely important aspect of sleep is the cycle between REM and NREM sleep. This pattern was first discovered in 1937 using EEG, and is very important to understanding of movement during sleep.

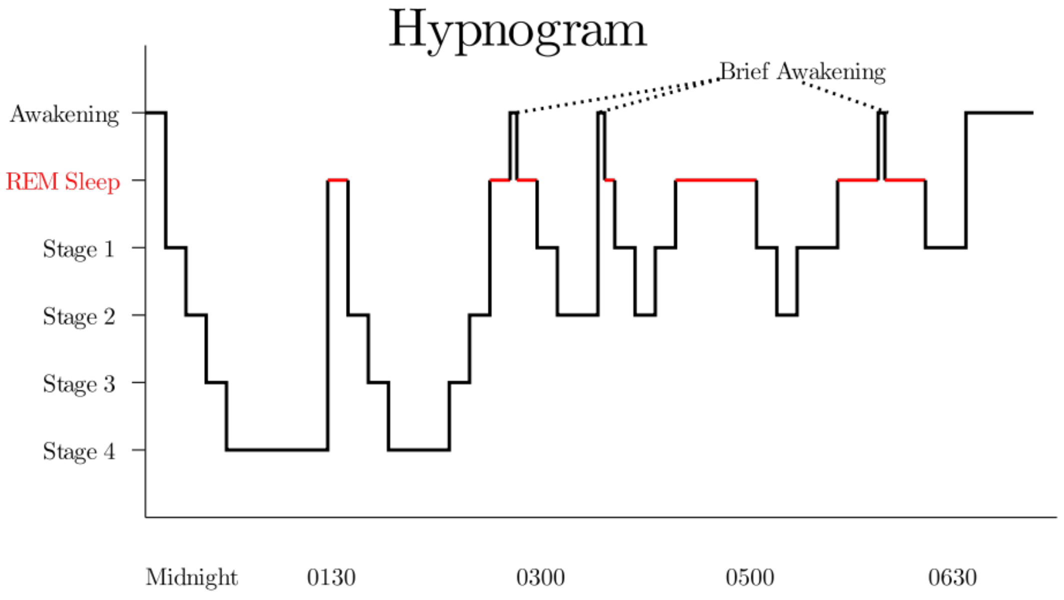


Figure 0: Normal sleep Hypnogram.

REM sleep has been shown (again via EEG) to line up with dreaming. Additionally, being denied REM sleep will lead to people being not very mentally refreshed and eventually falling right into REM sleep. However, it is not quite understood exactly what happens during REM sleep that causes it to be so important. For the purposes of this paper, however, REM sleep has been observed to be the time when most movement occurs while someone is sleeping. Because of this, we can expect our data to follow some sort of pattern similar to Figure 0. The major complicating factor for this study is the number of things that can influence a person's sleep cycle. Caffeine, alcohol, jet lag, anxiety, and stress are just a few of the factors that can cause variation from a person's baseline sleep pattern.

Data

In order to study a person's movement while sleeping, we need some sort of cheap, easy-to-use, low impact sensor in order to detect when people are moving. The approach used in this study is a smartphone's accelerometer. Leveraging the Android application "Sleep as Android," data was able to be collected across 3 subjects over about 3 months. The application works very simply. Start it up when you're ready to go to sleep, place it on your mattress near your body, and go to sleep. When you wake up, you disable the app, and it creates a record for that night's movement activities.

Collection Method Advantages

The benefits to using this application were numerous. Firstly, it was cheap, at only $3.00 past 30 days of use. Additionally, it uses a platform that most people already have on their nightstand, so no additional hardware needs to be added (purchased) to someone's room. Collection in a person's room with no new hardware involved has a secondary benefit of keeping a person relaxed. They may even forget entirely that their cell phone is on their bed, which means that there will be no affect by the device on their sleep patterns. If we were to study these movement patterns with something like EEG, which is only used in a hospital, we would have to find some way to correct for the fact that the person is likely to be more stressed due to not being in their home environment with an obtrusive device on their head.

Collection Method Disadvantages

Like all closed source off-the-shelf solutions, this one has problems. To date, the author has not published an interface document describing the application's output. The data we are using are defined as a "level" associated with a period of time. Testing and generic information from the author has led me to deduce that this is an accelerometer magnitude averaged out over 5 minutes. Noise also seems to be removed from the average. Unfortunately, there is no documentation or source code to back up this assertion, but testing with the application leads me to relatively high confidence with this conclusion. Additionally, not only is the data averaged, but also the time spacing is extremely high between averages. This is far less than ideal for our delay coordinate embedding techniques. This fact alone will make getting a solid, trustworthy estimate for the system's dimension, let alone a Lyapunov exponent, out of our data very difficult.

Results

For this study we had a total of 3 subjects. First to be presented will be the cross-subject aggregate results. Secondly will be the subject comparison results. One important note is that each "data set" that I refer to has a 1:1 relationship to one sleep movement recording by the application for one night, by one person.

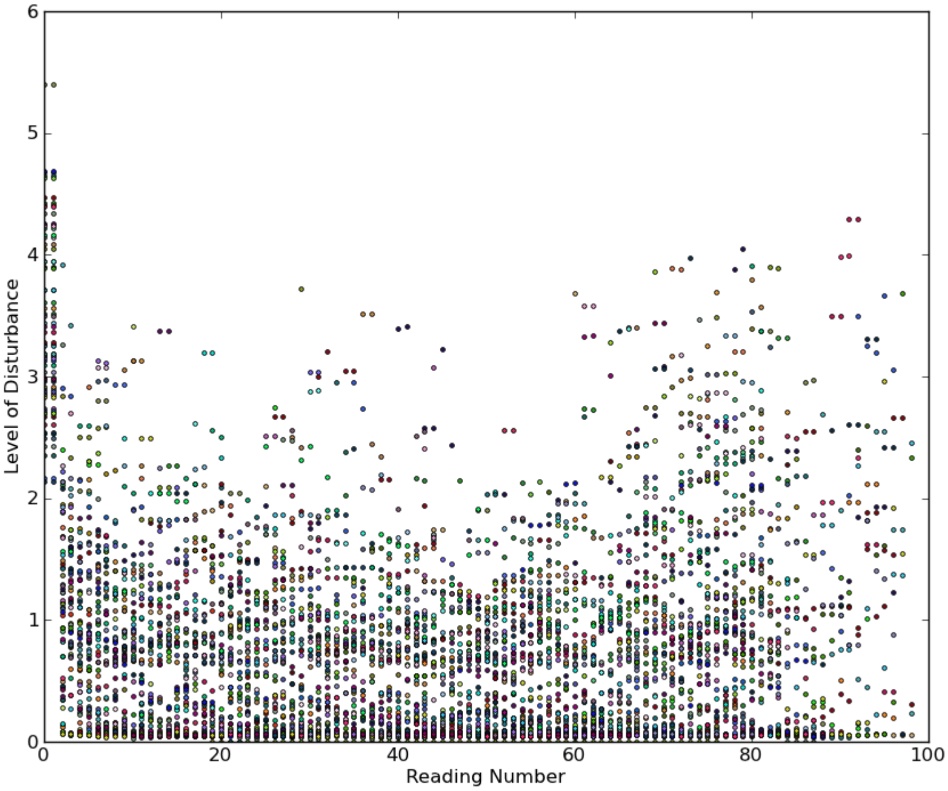


Figure 1: All raw reading data, across subjects.

Cross-Subject Results

First we needed to estimate a τ for our delay-coordinate embedding. Using Tisean's mutual function, we came up with a first minimum at about τ=3, globally across all of our data sets (see Fig 3.)

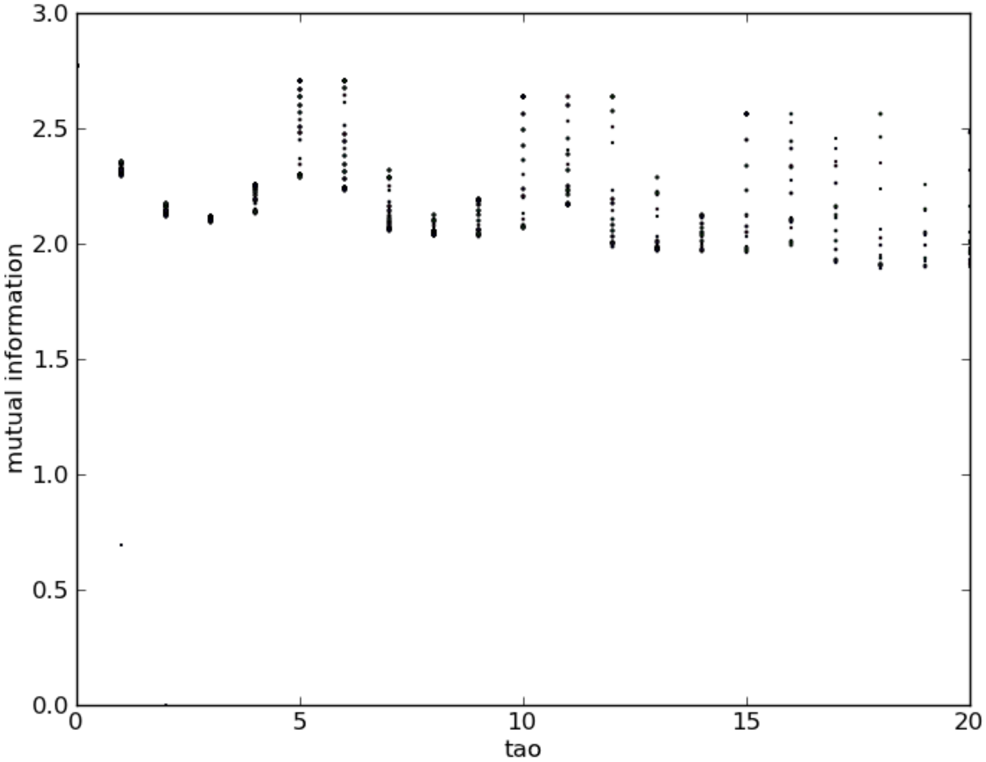


Figure 2: Mutual information across all datasets.

Next, we had to come up with some sort of dimensional analysis for our data sets. In this case, we are using Tisean's false_nearest tool. Our results were not nearly as clean in this case, as the shortness of the datasets really impacted our ability to check high dimensions before we ran out of points. Only about 25% of our datasets were able to find a ratio below 0.1 at any dimension (our criteria for calling it a "good" dimension estimate). Of these 25%, the average dimension was at about 7. This is the *m* value that I used for all the entire set of delay coordinate embedding from here on.

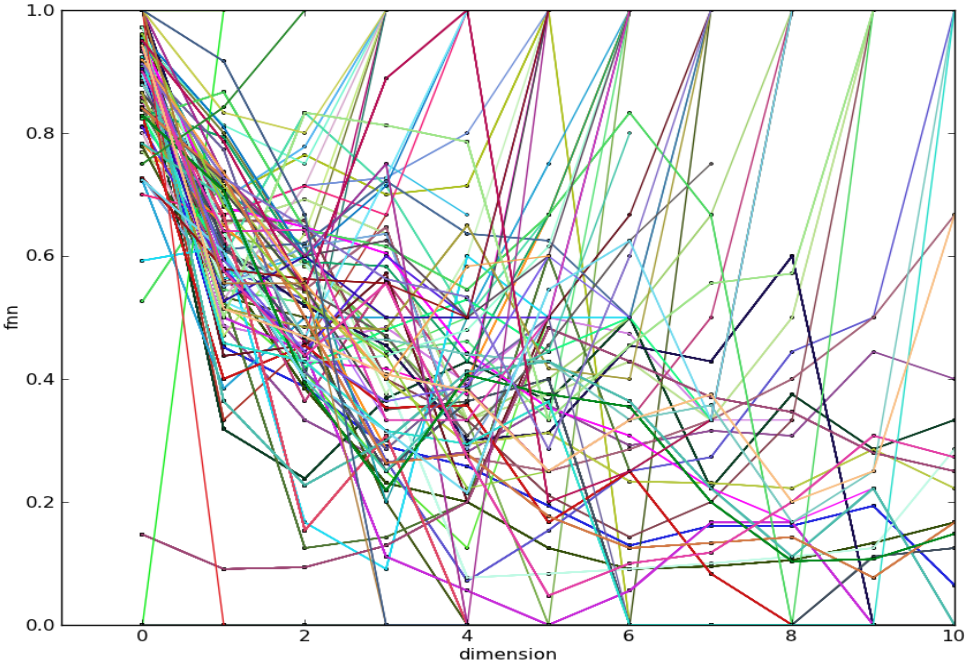


Figure 3: FNN by dimension. Goes to 1 if the dataset is no longer long enough to give a valid result.

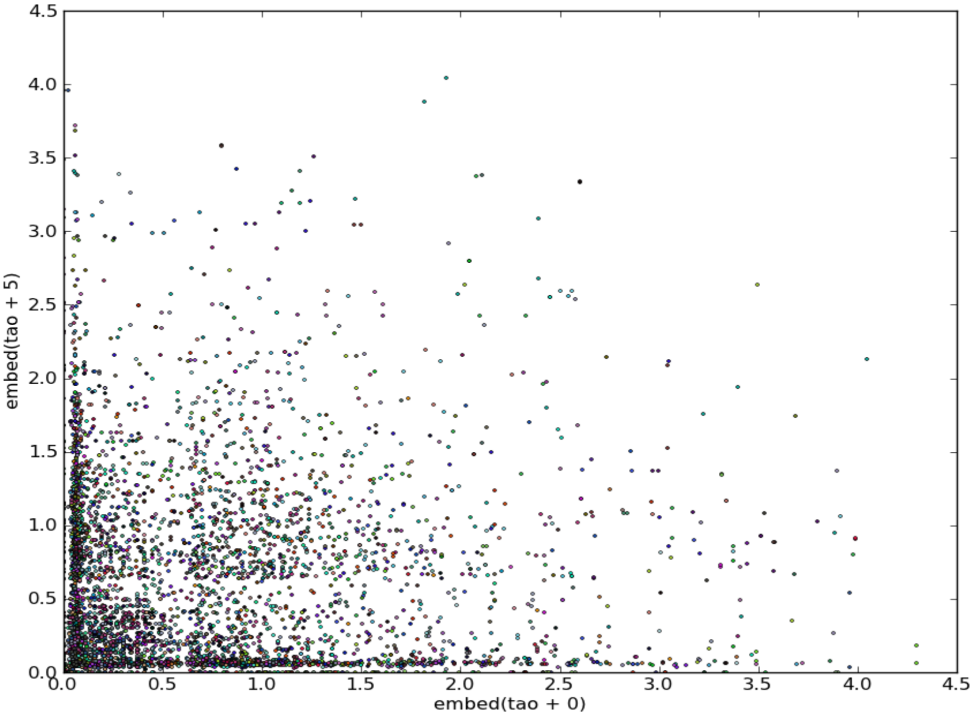


Figure 4: Embedding of all datasets, on an example projection.

All of this will allow me to create a plot of the local Lyapunov explonents in an attempt to find a scaling region where I can calculate a global Lyapunov exponent for the system. This plot is shown in Figure 5, and at least, we have something interesting.

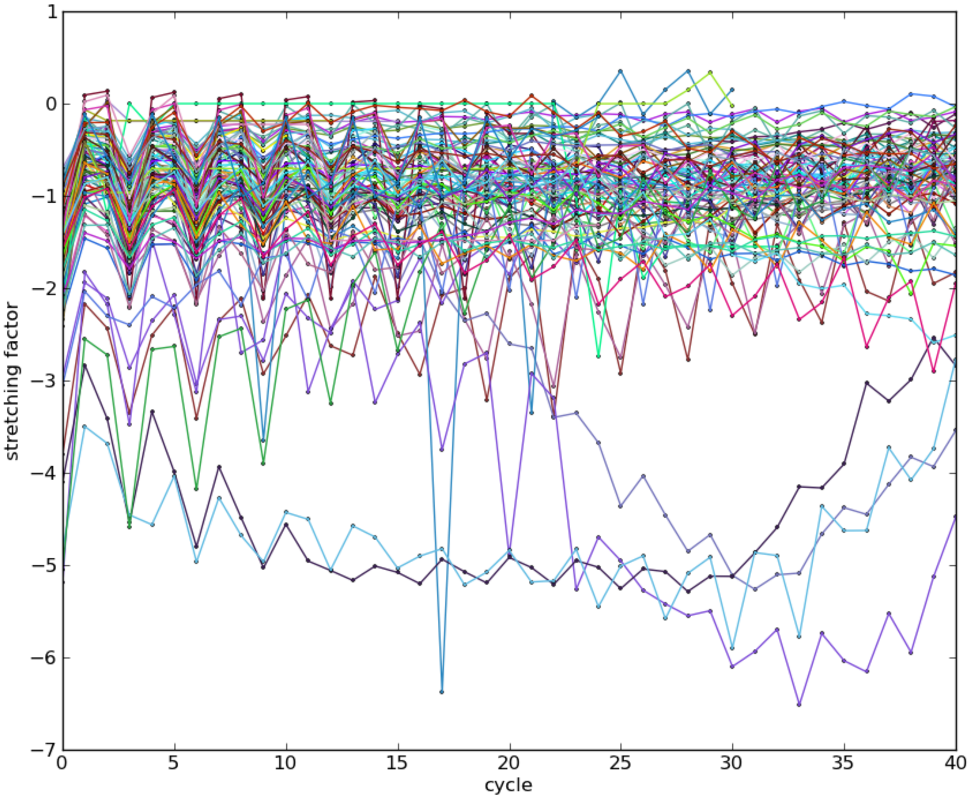


Figure 5: Lyapunov Exponent calculation on all datasets.

Subject Contrasts

Figures 6 and 7 show some lyap_r base plot differences between two different subjects. We see there is a high degree of similarity between two different people for what kind of local Lyapunov exponents we're getting.

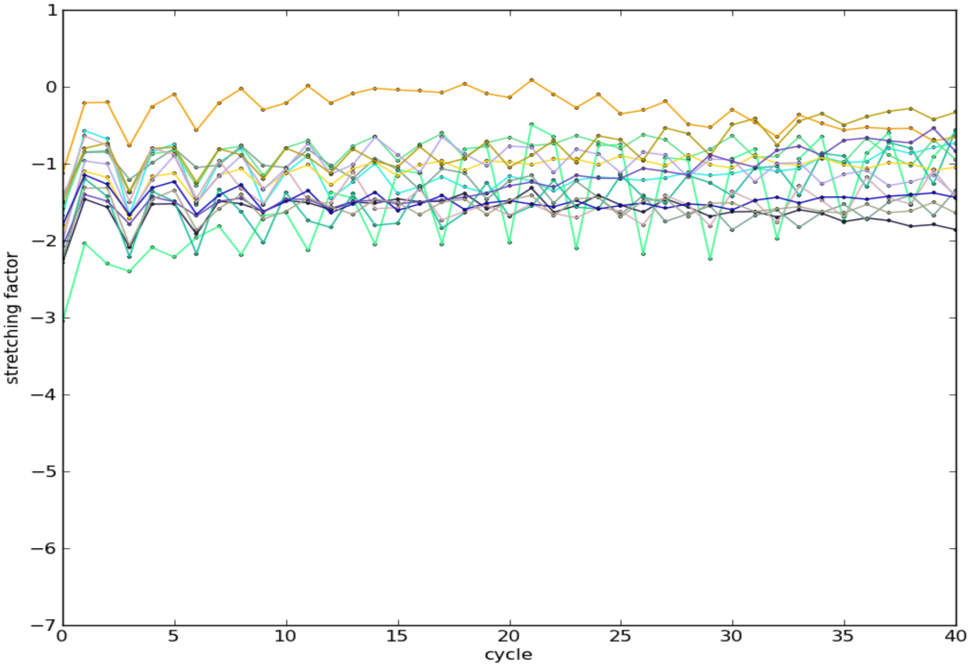


Figure 6: Subject 2 lyap_r output

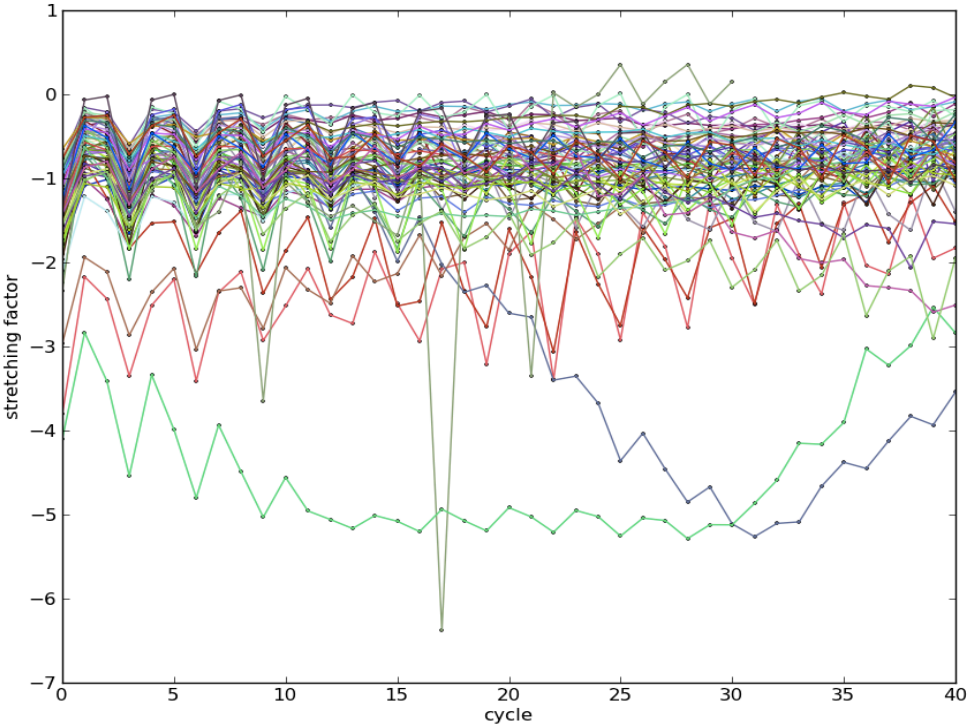


Figure 7: Subject 0 lyap_r output

While means between subjects seem relatively similar, I also explored potential causes for the outliers we see. For example, in Figure 7 there are two meandering lines that are well below the mean, as well as one trajectory with a massive negative spike. In Figure 6, we see a line that trends well above the mean. Figures 8 and 9 show some representative samples of the raw data from these outliers.

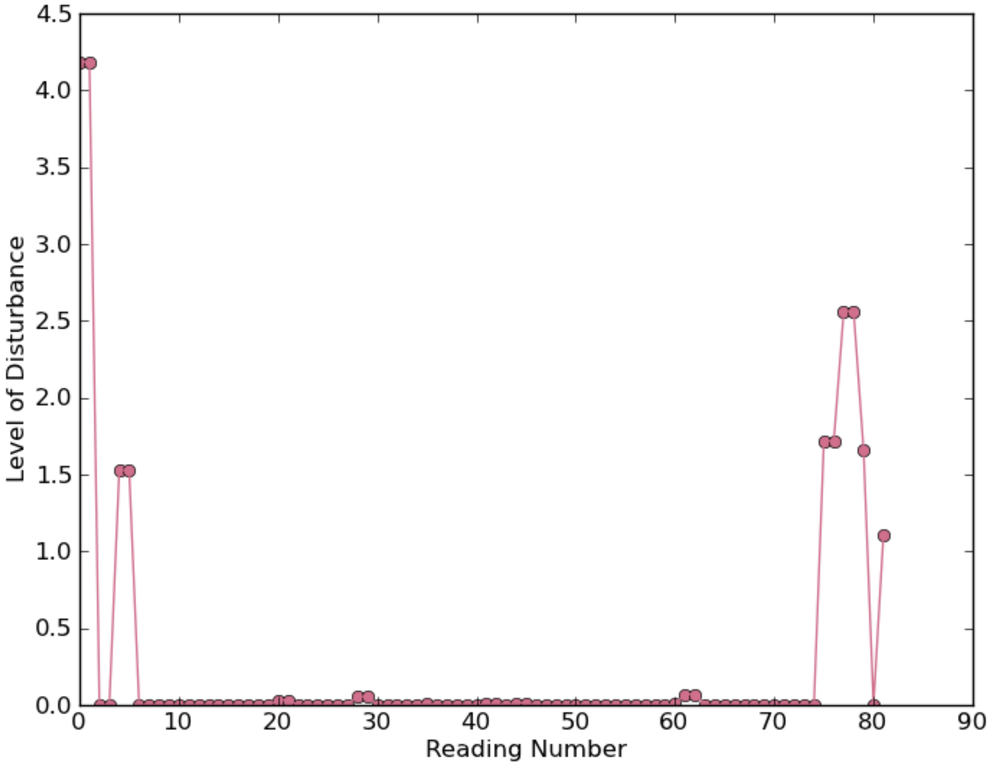


Figure 8: Below mean trending line on lyap_r's raw data.

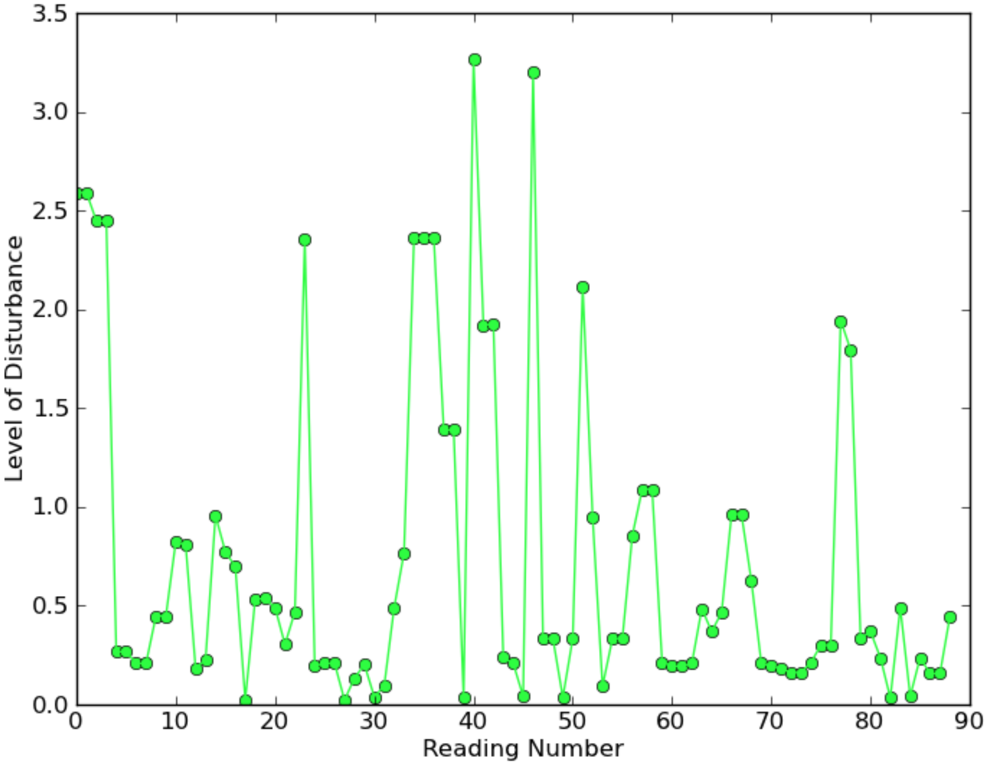


Figure 9: Above mean trending line on lyap_r's raw data.

Most of the outliers similar to Figure 8 are explainable as some form of bad data, like the phone falling off the bed, getting kicked in the night into a poor placement, or lying on the phone. Additionally, a night heavy with NREM sleep instead of REM sleep may also be responsible for some more moderate deviation below the mean. Lines like Figure 9, which are much less common, are considerably more interesting. These seem to show nights with high frequencies of REM sleep, or waking up frequently through the night. This could also be an indicator of REM Rebound, but I hesitate to draw any major conclusions due to lack of data.

Conclusions

With some of these outliers removed, we end up with a very nice figure in Figure 10, which shows what looks like the beginning of a scaling region, similar to what we've seen with data collected from a dampened pendulum. When I fit a line to this data, I was able to get Figure 11, with an equation of: y=0.004272x+(-1.010136). This slope shows a positive Lyapunov exponent, hinting at a chaotic attractor! However, I have a hard time putting a lot of faith in these numbers for a few reasons. Most importantly, our data is simply not very good. This study really requires considerably smaller delta t readings in order to be able to draw more believable conclusions. Secondly, the scaling region is not very well defined. If the data were longer, as well as more frequent, we may have been able to run more cycles, and perhaps draw a better-defined scaling region. Overall, I believe this is a good introductory study into a poorly understood part of people's every day lives.

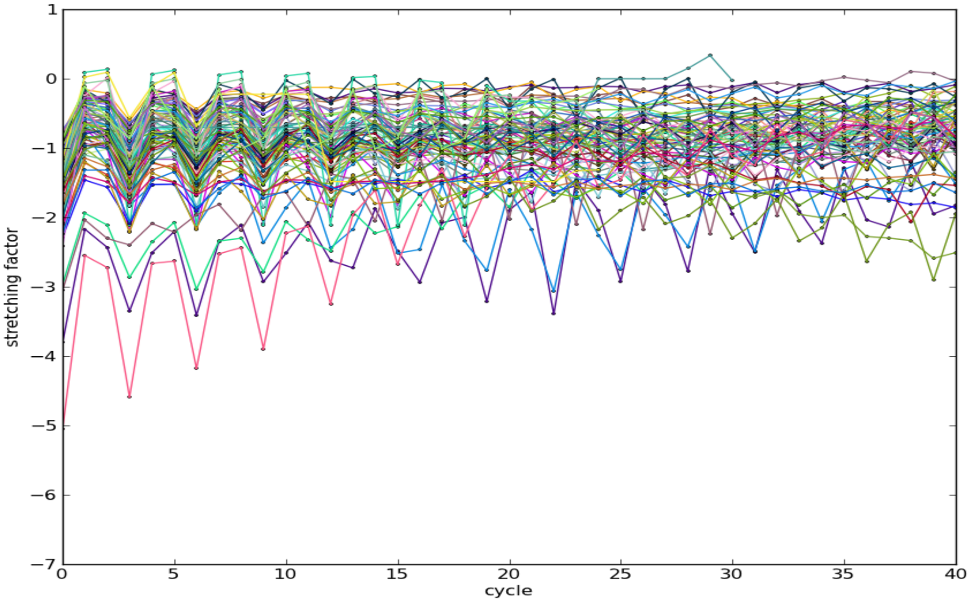


Figure 10: All subjects with most outliers removed.

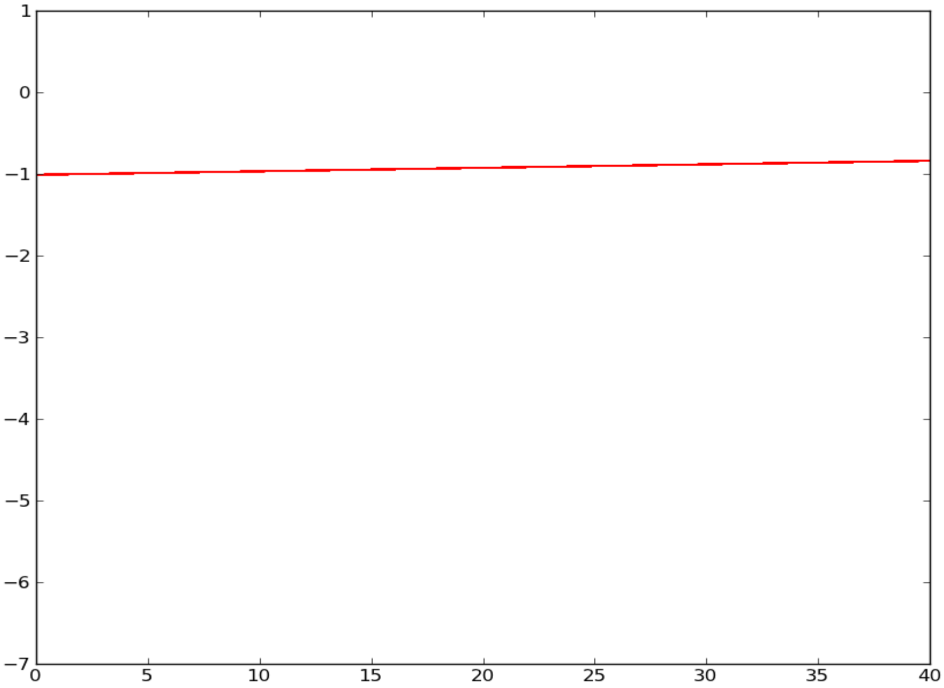


Figure 11: Line fit to Figure 10's data.

Further Work

A good continuation of this research would be to improve the data quality immensely. This can be done fairly easily by writing an open source application that does a better job of reading and understanding the individual accelerometers on people's cell phones. A repeat of this same analysis with better data would be a good next step.

Bibliography

A. Wolf, “Quantifying chaos with Lyapunov exponents,” in Chaos, Princeton University Press, 1986.

J. Fell et al., “The calculation of the first positive Lyapunov exponent in EEG data,” EEG and Clinical Neuropsychology: 19 March, 2003.

N. Pradhan and P.K. Sadasavian, “The nature of dominant Lyapunov exponent and attractor dimension curves of EEG in sleep,” Computers in Biology and Medicine: September 1996.

Sleep as Android. May 1, 2012, <https://sites.google.com/site/sleepasandroid/>

Public Library of Science. “The Human Brain Is On The Edge Of Chaos.” *ScienceDaily*, 19 Mar. 2009. Web. 1 May 2012

National Sleep Foundation. May 1, 2012 <http://www.sleepfoundation.org/>

National Insititute of Health. May 1, 2012 <http://www.nih.gov/>