Relatório do Modelo de Criptografia de Imagem por Transformação Caótica

Amanda Costa amanda.cs@usp.br
João Pedro de Freitas jpfreitas2001@usp.br
Juan Nogueira juan.nog@usp.br
Maria Rita Xavier mrxavier74@usp.br
Octavio Augusto Potalej oapotalej@usp.br
Samuel Garcez samueltgarcez@usp.br

Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo

O presente relatório investiga o conceito de Caos na matemática e suas aplicações nas ciências. Em particular, é aprofundado o estudo da transformação do gato de Arnold e, a partir de sua teoria, cria-se um método de criptografia de imagem, com detalhamento do trabalho computacional envolvido.

1 Introdução

A teoria do caos busca explorar sistemas determinísticos e, portanto, com lógica bem definida, mas que no curso de seu desenvolvimento, tomam complexidade tamanha que se assemelham a sistemas aleatórios. Segundo Laplace, em seu livro "Ensaio filosófico sobre as probabilidades", o próprio universo, com sua complexidade e aparentemente aleatoriedade, seria um sistema caótico e (consequentemente) determinístico.

Ao longo do relatório, detalharemos a sua aplicação na criptografia de imagens (criptografia caótica), em específico, através da transformação do gato de Arnold, que incide no fato de que operações simples em um espaço vetorial levam a um sistema caótico cuja descoberta da configuração inicial de seus vetores se torna uma tarefa não trivial.

2 O Modelo

2.1 A Definição de Caos

Caos é o nome popular dado aos sistemas dinâmicos não lineares. De acordo com Alves (2011), a Teoria do Caos é uma nova forma de analisar a complexidade da natureza e das organizações. Com ela, é possível definir quais parâmetros e condições causam modificações e os comportamentos de alguns sistemas. Ainda, de maneira simplificada, a sua ideia central está no fato de que uma pequena mudança nas variáveis de um sistema pode causar impactos complexos, desconhecidos e de grandes proporções no futuro.

No início dos anos 1960, o cientista Edward Lorentz foi o expoente do estudo do caos. Com o seu trabalho sobre o efeito borboleta, ele descobriu que uma pequena mudança no par de variáveis que regem um sistema produziam efeitos desproporcionais. Em um curto período de tempo não seria possível notar essas alterações. No entanto, em uma semana ou um mês os efeitos seriam completamente diferentes. Assim, o efeito borboleta seria a quebra de ideia de que uma certa mudança gera resultados proporcionais.

Além disso, o caos pode gerar gráficos de geometria complexa que podem indicar quando um evento é caótico - ou não - e suas predições. Através do movimento de um objeto em um sistema dinâmico, é possível plotar um gráfico abstrato contendo todas as informações do objeto. Esses gráficos são denominados atratores e podem ser de ponto imóvel (as variáveis não mudam, assim, não há mudança no movimento do objeto - ele permanece estático), de movimento previsível (quando o caminho a ser percorrido pelo objeto é conhecido e apenas é necessário saber a velocidade com a qual ele se movimenta) e de caos total (as equações são imprevisíveis e geram fractais - gráficos de geometria complexa e detalhes infinitos).

Assim, como mostrado pela revista Superinteressante (2020), os estudos do caos ampliaram-se muito após a descoberta de Lorentz. Tanto que, atualmente, cientistas afirmam que a sua manifestação é percebida de muitas formas na natureza e na nossa vida cotidiana.

2.2 Aplicações da Teoria do Caos

A teoria do caos possui potencial para relacionar e analisar as interações complexas entre os sistemas e os ambientes nos quais funcionam. Ela é útil para a predição do comportamento desses sistemas a curto e a longo prazo. Sendo assim, percebe-se que muitas podem ser as suas aplicações nos mais distintos setores de conhecimento. Por ora, serão apenas apresentadas aplicabilidades na Biologia e na Engenharia.

De acordo com Cury (2012), os gráficos de recorrência, uma técnica relacionada a teoria do caos que mostra o comportamento de sistemas complexos, é amplamente usada na Biologia para a análise de batimentos cardíacos de indivíduos. Esses gráficos são gerados através de resultados dos pacientes. Pessoas que resultavam em um gráfico com coloração mais densa e escura sinalizavam um alto risco de problemas cardíacos. Por outro lado, o diagrama de cor menos densa e clara mostrava que esses pacientes possuíam um baixo risco de sofrer com doenças cardíacas. Desse modo, infere-se a contribuição do caos para a Biologia.

Diferentemente da Biologia, o caos é aplicado na Engenharia através da técnica da entropia, que é usada como os dados para plotagem dos gráficos de recorrência. Nos gráficos, são comparadas informações de arquiteturas antigas com edificações que se deseja realizar a manutenção. Com essa análise, é possível determinar se uma estrutura está boa, ruim ou se pode apresentar mudanças no futuro.

Em suma, infere-se que a teoria do caos pode ser aplicada em diversas áreas. Na Biologia, ela é útil na predição de doenças cardíacas em indivíduos. Já na Engenharia, ela ajuda na análise de informações estruturais de construções, o que também pode evitar prejuízos futuros. Todas essas aplicações acontecem através do estudo de gráficos de recorrência, uma técnica da caótica.

2.3 A transformação do Gato de Arnold

A transformação do gato de arnold é dada por Γ: $R^2 \rightarrow S$ tal que $S = (x, y) : 0 \leq x < 1, 0 \leq y < 1$ e $\Gamma(x, y) = (x + y, x + 2y) \bmod 1$. Observe que a transformação de Arnold utiliza da aritmética modular para transformar pares ordenados de números reais em pontos de S. Ou seja, a aplicação de mod 1 na transformação garante que, independente do momento em que seja feita, a imagem que foi deformada para fora do quadrado S retorne aos limites do quadrado mantendo suas posições. Em notação matricial podemos descrever a transformação da seguinte forma:

$$\Gamma\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \bmod 1$$

Ou, fatorando:

$$\Gamma\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \bmod 1$$

O fenômeno a ser observado na transformação acima ocorre quando esta é iterada n vezes. A transformação, depois de n iterações de um ponto $p = (x_0, y_0)$ resulta em q, tal que $q = (x_0, y_0)$. Dito isso, constrói-se sobre n o conceito de periodicidade. No entanto, por conta da aritmética modular não poder ser utilizada como parte de uma transformação, pode-se utilizar do conceito de "plano ladrilhado" para se obter resultados semelhantes. Assim, a dita imagem se repete a cada unidade de x e y.

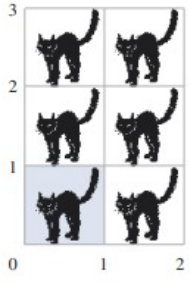

Fig. 1: Imagem no plano ladrilhado

Dessa maneira é possível analisar a transformação de Arnold como a seguinte operação:

$$\Gamma\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Cada ponto do domínio da transformação tem uma periodicidade definida, ou seja, um valor de n, de maneira que a cada n iterações, encontramos o mesmo resultado (igual a situação original) para um par ordenado. Assim, para encontrar o valor de n em que todos os pares ordenados do domínio retornem ao resultado original simultaneamente, basta encontrar o mínimo múltiplo comum dos valores de n de todos os pares ordenados do domínio considerado.Vale lembrar que a periodicidade utilizando diferentes abordagens também apresenta características diferentes.

Utilizando-se da aritmética modular, um ponto p qualquer volta a sua posição inicial após n iterações. Em contraste, utilizando-se do ladrilhamento a periodicidade

existe de forma que o lugar de cada ponto que é substituído recebe outro de mesma cor.

2.4 Aplicação da Transformação do Gato de Arnold: Imagens

Utilizando uma imagem quadrada com p pixels em seus lados, ou seja, p^2 pixels no total, é possível observar uma aplicação do Gato de Arnold com clareza. Dessa maneira, estabelecendo como m e n índices, todo ponto-pixel q do quadrado pode ser definido como $q = (\frac{m}{p}, \frac{n}{p})$. A figura abaixo demonstra isto para o caso de $p = 101$.

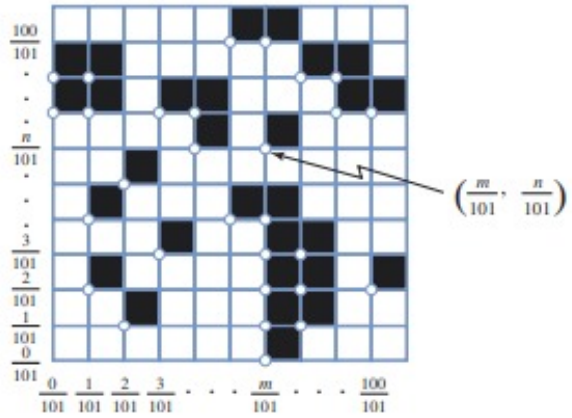

Fig. 2: Imagem dividida em pontos-píxels.

Tal que esses pixels tenham lado $1/p$ e m, n estão no intervalo $[0, p-1]$.

Cada pixel recebe uma cor específica para a geração da imagem, o que recebe o nome de aplicação de píxels. Aplicando a transformação do Gato de Arnold nos pares ordenados de pontos de píxel, obtemos a seguinte expressão:

$$\Gamma\left(\begin{bmatrix}\frac{m}{p}\\ \frac{n}{p}\end{bmatrix}\right) = \begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}\frac{m}{p}\\ \frac{n}{p}\end{bmatrix} \bmod 1 = \begin{bmatrix}\frac{m+n}{p}\\ \frac{m+2n}{p}\end{bmatrix} \bmod 1$$

2.5 Definição de uma Transformação Caótica

Uma aplicação Γ de um conjunto S sobre si mesmo é dita caótica se satisfizer as condições:

(i) S contiver algum conjunto denso de pontos periódicos de Γ

(ii) e existir algum ponto em S cujas iteradas por Γ são densas em S.

Em que os pontos racionais são periódicos e densos em S. Neste relatório, são utilizados apenas valores racionais para os pontos de pixel.

3 O Programa

O programa construído para os cálculos apresentados e realizados neste documento é dividido em classes e é de propósito geral em sua maior parte, ou seja, seus métodos normalmente são utilizados com mais de um propósito.

3.1 Função Gama

Primeiramente, o método principal está contido na classe `AlgoritmoGeral` e calcula a `funcaoGama`, tendo como parâmetros a razão da posição cartesiana de um pixel (x e y) pela quantidade de pixels, além da chave C a ser utilizada, e é calculada da forma que foi apresentada:

$$\Gamma(v) = Cv \bmod 1$$

onde v é o vetor $\begin{bmatrix}\frac{x_p}{n}\\ \frac{y_p}{n}\end{bmatrix}$, sendo n a quantidade de pixels na tela.

```
def funcaoGama(x, y, C):
    prod = Matrizes.produtoMatriz(C, [[x], [y]])
    return Matrizes.mod1Matriz(prod)
```

Os métodos `produtoMatriz` e `mod1Matriz` da classe `Matrizes` calculam, respectivamente, o produto entre duas matrizes e o mod 1 de um vetor como definido anteriormente, sendo aplicado em ambos os elementos do vetor.

Para que todos os pixels de uma imagem passem pela transformação realizada pela função Γ, o método `iterarMatriz`, também da classe `AlgoritmoGeral`, faz a iteração quantas vezes for requisitado. Seus parâmetros são:

- A chave C;
- A matriz que contém a informação da imagem;
- O limite de iterações (para evitar sobrecargas);
- Se se deseja salvar todas as iterações na forma de imagem (opcional); e
- Se o limite de iterações encerra como fracasso ou retorna o que foi calculado até então (opcional).

Antes de tudo, se armazena a matriz original na variável `matrizBase` e em seguida se a insere em uma lista. Esta lista conterá todas as iterações efetuadas durante o cálculo. Há também o contador, que começa em zero e encerrará o loop de iterações assim que atingir o limite passado como parâmetro, a menos que se encerre a função no caminho.

```
def iterarMatriz(C, matriz, limIt, salvar = False,
↪  limItEncerra = False):
    matrizBase = matriz
    listaIteracoes = [matrizBase]
    contador = 0
    while contador < limIt:
```

Dentro desse ciclo **while**, há dois loops: um que gera uma matriz vazia e o outro que transforma a matriz atual pela função Γ:

```
while contador < limIt:
    contador += 1
    # matriz vazia
    lista = [ ]
        for i in range(len(matriz)):
            l = [ ]
            for j in range(len(matriz)):
                l.append(0)
            lista.append(l)
    # transformação
    for i in range(len(matriz)):
        for j in range(len(matriz)):
            gama = funcaoGama(i/len(matriz),
            ↪  j/len(matriz), C)
            x = round(gama[0][0]*len(matriz))
            y = round(gama[1][0]*len(matriz))
            lista[i][j] = matriz[x][y]
    matriz = lista
    listaIteracoes.append(lista)
```

Ao final de cada iteração, são realizadas duas verificações. Primeiro, se se deseja salvar as iterações, e nesse caso a matriz transformada é convertida para imagem (como será detalhado mais à frente). Segundo, se a matriz transformada é igual a original e se se objetiva encerrar as iterações neste caso (caso que facilita o reconhecimento do período de uma chave quanto a uma imagem).

3.2 Imagens Armazenadas em Arquivos .txt

Para maior precisão no momento de se transmitir uma imagem, é natural que esta seja convertida inteiramente para uma matriz e armazenada de alguma forma. Neste caso, a matriz pode ser armazenada em um arquivo do tipo .txt, o que facilita seu envio e compartilhamento. A linguagem de programação utilizada (Python) facilita esse processo.

Para ler ou escrever um arquivo .txt em Python se utiliza a função nativa `open()`. No caso atual, para a leitura, há uma classe `Arquivos` que possui um método `matrizDeArquivo` elaborado da seguinte forma:

```
def matrizDeArquivo(nome):
    w = open(f'./{nome}.txt', 'r')
    txt = w.read()
    m1 = txt.split('\n')
    matriz = []
    for m in m1:
        linha = m.split(' ')
        listc = []
        for l in linha:
            if l != "": listc.append(float(l))
        matriz.append(listc)
    matriz.pop(-1)
    w.close()
    return matriz
```

A formatação da saída da leitura de um arquivo depende exclusivamente do formato em que o arquivo foi escrito. Neste caso, é considerado que cada linha da matriz está dividida por uma quebra de linha (`\n`) e que os elementos de uma linha estão divididos por um espaço. Porém, esta disposição varia e a formatação deve ser alterada conforme a disposição.

Escrever um arquivo do tipo .txt é tão simples quanto ler, e da formatação vale o mesmo que para a leitura:

```
def salvarEmTxt(matriz, nome, dir):
    # matriz que conterá todas as linhas juntas
    txt = ""
    for linha in matriz:
        for coluna in linha:
            txt += f"{coluna}  "
        txt += '\n'
    # agora, resta salvar
    f = open(f'{dir}/{nome}.txt', 'w')
    f.write(txt)
    f.close()
```

3.3 Leitura e Gravação de Imagem

O trabalho com imagens exige a utilização de duas bibliotecas do Python nomeadas *Pillow* e *Numpy*, dedicadas a operar com arquivos desse tipo e a trabalhar com cálculos, respectivamente. Para ser feita a leitura e transformação em uma matriz, são necessários poucos comandos:

```
from PIL import Image
from numpy import asarray

def lerImagem(dir):
    img = Image.open(dir)
    matriz = asarray(img)
    return matriz
```

A variável `matriz` armazena, na realidade como um tensor, a cor de cada pixel da imagem passada como parâmetro no formato *RGB* ou *RGBa* (se houver opacidade).

Para construir uma imagem a partir de uma matriz, dessa forma, é necessário possuir uma lista com as informações de cada pixel a ser integrado em uma imagem:

```
from PIL import Image
from numpy import asarray

def desenhar(R, G, B, nome, tamanho):
    [Comprimento, Altura] = tamanho
    arquivoNome = (nome, "PNG")
    obj = Image.new("RGB", (Comprimento, Altura))
    setar = obj.load()
    for linha in range(Comprimento):
        for coluna in range(Altura):
            setar[linha, coluna] = (R, G, B)
    obj.save(*arquivoNome)
```

3.4 Exemplo

Dada a seguinte imagem, devidamente armazenada em três arquivos que contém seus tons vermelhos, verdes e azuis (RGB):

Fig. 3: Imagem inicial

Aplicando a transformação do gato de Arnold algumas vezes se utilizando da chave padrão $C = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$, é obtida uma imagem interessante.

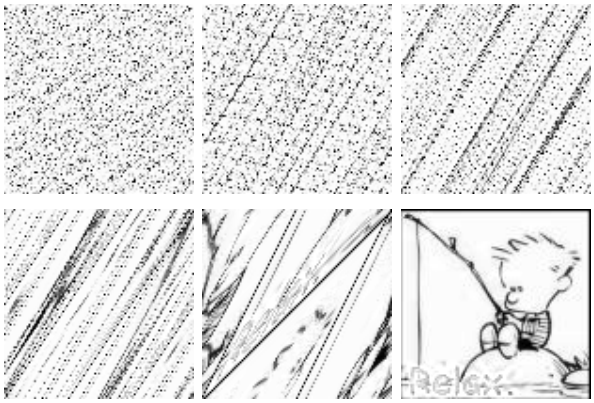

Fig. 4: Transformações aplicadas iteradamente.

É importante observar que no caso da descriptografia de uma imagem transformada, se faz suficiente transformar tão somente apenas uma das cores e apenas depois juntar todas, pois isto reduz o trabalho computacional necessário. Assim, tem-se a seguinte imagem, após juntar as três cores:

Fig. 5: Imagem original colorizada

4 Descriptografando

No exemplo apresentado anteriormente, a situação de descriptografar uma imagem era simples, pois se tinha a chave utilizada e a quantidade de transformações necessárias era baixa. Porém, no caso de não se ter uma chave e tampouco uma estimativa da quantidade de transformações necessárias, descriptografar uma imagem é uma tarefa trabalhosa e que demanda muito poder computacional, como será visto a seguir no processo de descriptografar a seguinte imagem:

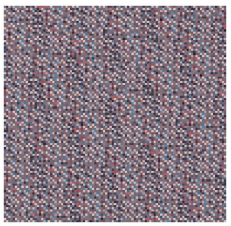

Fig. 6: Imagem transformada

4.1 Chaves Possíveis e Filtro de Imagens

A quantidade de chaves possíveis pode ser calculada dado que se possui uma limitação estabelecida: seus valores devem ser inteiros entre 0 e 100. Ademais, a natureza da transformação do Gato de Arnold impõe que a chave deve ter determinante 1 e os módulos de seus autovalores devem ser diferentes de 1, reduzindo o número de combinações possíveis para 11.974.

Este número, porém, é ainda muito grande, pois considerando que se calcule 55 transformações para cada chave (número estimado se considerando que algumas chaves possuem período 5 e outras maior que 100), seriam geradas cerca de 660 mil imagens, e como se faz necessário olhar cada uma, o processo é demasiado demorado e custoso.

Foi observado que algumas imagens possuíam silhuetas e estranhas repetições de pequeninas outras imagens, como nas duas a seguir:

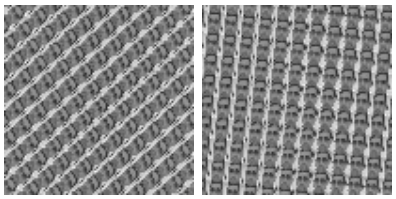

Fig. 7: Pequenas silhuetas podem ser vistas de forma repetida.

O que se difere de imagens que contém apenas algumas linhas, e então, para evitar a geração das 660 mil imagens, foi utilizada uma biblioteca em Python chamada *imgcompare* que, se utilizando da já mencionada *Pillow*, compara cada pixel entre duas imagens e retorna algumas informações, como o percentual de diferença entre ambas, visando salvar apenas imagens diferenciadas como as apresentadas anteriormente:

```
import imgcompare

def comparar(img1, img2):
    dif = imgcompare.image_diff_percent(img1, img2)
    return dif
```

Esta diferença, no entanto, é extremamente sensível, dado a sua forma de ser calculada. Cálculos posteriores indicam que a diferença entre a seguinte imagem e a imagem que se busca é de cerca de 20,34%, embora ambas sejam visualmente bastante diferentes (como se poderá ver mais a diante)

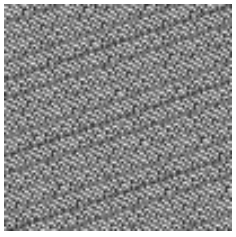

Fig. 8: Imagem de base

4.2 Ataques por Força Bruta

Se tendo as combinações e este pequeno filtro, foi-se testando tantas chaves quanto possível em duas máquinas, atingindo-se por volta de 8 mil chaves testadas durante um período de cerca de 8 horas no total.

Cada chave era selecionada aleatoriamente e retirada para que não pudesse ser selecionada novamente. Sortear foi uma saída encontrada para evitar de se despender muito tempo com as chaves mais iniciais e simples, já que a chave utilizada na encriptação foi escolhida por um humano e a intenção na escolha era a de se complicar a decriptação.

Se percebeu após todos os 8 mil testes, porém, que o filtro não estava sendo preciso o suficiente. Por falta de testes anteriores que pudessem inferir estatisticamente um bom valor de filtragem, se utilizou 20,5%, e visto que após análise posterior foi descoberto que a chave correta havia sido testada, a imagem correta foi eliminada pelo filtro.

Não se podendo dedicar novamente todo o tempo gasto para o ataque por força bruta realizado, foi necessário requerer alguma informação que pudesse reduzir ao menos um pouco a quantidade de combinações. O informado foi de que a chave era simétrica.

4.3 Simetria da Chave e a Solução

Uma matriz ser simétrica significa que sua transposta é igual a própria matriz, ou seja, seja C a chave, tem-se:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \end{bmatrix}$$

Ou seja, $c_{21} = c_{12}$, e apenas 94 das cerca de 12 mil chaves possíveis cumprem com esse requisito. Foi realizado novamente um ataque por força bruta nas 94 chaves encontradas, mas dessa vez sem o filtro, se gerando cerca de 5,7 mil imagens apenas, e no meio delas foi encontrada a correta com a chave $C = \begin{bmatrix} 85 & 47 \\ 47 & 26 \end{bmatrix}$, na 45ª transformação*:

Fig. 9: Tons vermelhos descriptografados e normalizados

Sendo a imagem colorizada a seguinte:

*Faz-se necessário observar que o defeito na imagem veio da origem da encriptação (informação confirmada com o grupo que a encriptou), bem como a responsabilidade da escolha da imagem também é da origem. O apresentado buscou somente apresentar a decriptação, conforme requerido.

Fig. 10: Imagem final e colorizada

Os programas podem ser encontrados *neste repositório*. O arquivo *gerarImagem.py*, quando rodado, permite se inserir um link de alguma imagem, cuja qual será redimensionada, além da chave, o número de transformações e algumas outras opções. Neste pode-se ver de forma prática o algoritmo em funcionamento.

4.4 Do algoritmo utilizado

4.4.1 Da transformação

É notável que o tempo demandado pelo algoritmo utilizado na decriptação é quase que totalmente destinado a calcular a transformação em cada pixel. A operação da forma que foi escrita necessita do produto de matrizes, o que é algo custoso, principalmente no caso de se ter de calcular tantas vezes quanto o número de pixels.

Um jeito mais elaborado e planejado de se calcular a transformação seria se utilizando da diagonalização da matriz chave no formato:

$$C = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} \lambda_u & 0 \\ 0 & \lambda_v \end{bmatrix} \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix}^{-1}$$

Onde u e v são autovetores de C e λ_u e λ_v são seus respectivos autovalores. Esta transformação otimiza a função no sentido de que para uma quantidade n de iterações, a transformação do Gato de Arnold necessita de calcular C^n, que em sua forma bruta é um grande produto (lento) e através da diagonalização se torna tão simples quanto λ_u^n e λ_v^n (rápido):

$$C^n = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} \lambda_u^n & 0 \\ 0 & \lambda_v^n \end{bmatrix} \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix}^{-1}$$

Dessa forma, se desenvolvendo a inversa e multiplicando as matrizes, tem-se:

$$C^n = \frac{1}{u_x v_y - v_x u_y} \begin{bmatrix} u_x v_y \lambda_1^n - u_y v_x \lambda_2^n & u_x v_x (\lambda_2^n - \lambda_1^n) \\ u_y v_y (\lambda_1^n - \lambda_2^n) & u_x v_y \lambda_2^n - u_y v_x \lambda_1^n \end{bmatrix}$$

O que acaba sendo mais ágil para valores de n suficientemente expressivos. Esta forma de cálculo, no entanto, não foi utilizada devido a certos problemas de precisão para valores de n acima de 30 que poderiam ser resolvidos, mas demandariam considerável tempo, o que não se tinha em demasia.

4.4.2 Da análise de imagens

Outro fator que dificultou a busca pela chave e imagem corretas foi a imensa quantidade de testes necessários e consequentemente de imagens geradas, cuja quase que absoluta parte nada continha de relevante. Foi mencionada a biblioteca *imgcompare* utilizada, que fez um filtro bastante considerável, mas demasiado sensível para diferenciar concretamente duas imagens 101×101 em escala de cinza.

Uma possível solução para um filtro melhor seria o uso de Redes Neurais Convolucionais (RNC), utilizadas geralmente no reconhecimento e processamento de imagens, mas que possuem propósito geral. Estas trabalham com tensores semelhante ao que se obtém se utilizando a biblioteca *Numpy*, como apresentado anteriormente, e poderiam ser aplicadas neste caso identificando imagens insuficientemente diferentes das que aparecem geralmente.

Esta solução se diferenciaria da utilizada, pois a *imgcompare* calcula a diferença entre duas imagens se baseando no histograma de ambas, e como uma imagem misturada é apenas a mesma imagem com pixels posicionados de forma diferente, a filtragem realizada se faz muito pouco produtiva, enquanto que uma RNC poderia compreender silhuetas e identificar na imagem, de fato, algo mais que apenas a posição de suas cores.

5 Conclusão e Considerações Finais

Sistemas caóticos usualmente são sinônimos de incerteza e imprevisibilidade, e assim o são se não for conhecida a sua configuração inicial. Caso seja, como por construção, podemos fazer uso da sua imprevisibilidade para evitar que qualquer um que não conheça essa configuração encontre dificuldades gigantescas para a descobrir.

O relatório apresentou um modelo que realiza este feito com sucesso, constituindo-se de uma das muitas aplicações da teoria do caos. Com a ajuda da álgebra linear e de ferramentas computacionais, tornou-se possível geometrizar e esconder a informação inerente a uma imagem ao mesmo tempo guardando a chave para a recuperar. A eficiência desse recurso, por sua vez, está garantida em vista da

dificuldade em descriptografar na ausência de melhores informações sobre a chave, não havendo outro método senão tentativa e verificação de milhares de imagens.

Variações do mesmo tipo de criptografia com outros tipos de informação, que fogem do escopo deste relatório, também seriam possíveis, bastando-se então relacionar a informação a um espaço vetorial e então o distorcer de forma caótica, com transformações arbitrariamente escolhidas.

Referências

[1] H. Anton and C. Rorres, *Álgebra Linear com Aplicações*. bookman, 2012.

[2] C. A. ALVES, "Teoria do caos e as organizações," 2011.

[3] R. Superinteressante, "O que é a teoria do caos?" 2020. [Online]. Available: https://super.abril.com.br/mundo-estranho/o-que-e-a-teoria-do-caos/

[4] D. M. Curry, "Practical application of chaos theory to systems engineering," 2012.