Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Metaheurísticas no Projeto de Autômatos Celulares para a Geração de Chaves em Criptografia de Fluxo

André de Souza Brito

JUIZ DE FORA
NOVEMBRO, 2017

Metaheurísticas no Projeto de Autômatos Celulares para a Geração de Chaves em Criptografia de Fluxo

André de Souza Brito

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Saulo Moraes Villela

Coorientador: Stênio Sã Rosário Furtado Soares

JUIZ DE FORA

NOVEMBRO, 2017

Metaheurísticas no Projeto de Autômatos Celulares para a Geração de Chaves em Criptografia de Fluxo

André de Souza Brito

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Saulo Moraes Villela
Doutor em Engenharia de Sistemas e Computação

Stênio Sã Rosário Furtado Soares
Doutor em Computação

Heder Soares Bernardino
Doutor em Modelagem Computacional

Lorenza Leão Oliveira Moreno
Doutora em Informática

JUIZ DE FORA
29 DE NOVEMBRO, 2017

Resumo

A preocupação com a segurança e a integridade de mensagens sigilosas sempre foi motivo de estudo. Uma das técnicas mais utilizadas com esse fim é a criptografia, que consiste em esconder uma mensagem tornando-a incompreensível, de forma a garantir que somente o remetente e o destinatário tenham acesso ao seu conteúdo original. Nesse processo o algoritmo responsável aplica sobre a mensagem uma chave, que tem como objetivo esconder a mensagem para envio ou revelar a mesma ao chegar no destino. Com isso, fica evidente que o sucesso de um algoritmo desse tipo depende diretamente da natureza da chave utilizada pelo mesmo. Muito do esforço usado na criação desses sistemas é direcionado para a tarefa de garantir que a chave usada assegure a segurança dos dados que foram criptografados. Nesse trabalho é apresentada uma solução sobre esse domínio baseada no uso de metaheurísticas para o projeto de autômatos celulares (ACs) unidimensionais caóticos usados como mecanismos geradores de chaves para algoritmos criptográficos de fluxo.

Palavras-chave: Criptografia de Fluxo, Autômatos Celulares, Resfriamento Simulado, Busca Tabu, Metaheurísticas.

Abstract

The concern about security and integrity of confidential messages has always been target of study. One of the most used techniques for this purpose is the encryption, which consists of hiding a message making it incomprehensible, ensuring that only the sender and the addressee have access to its original content. In this process, the responsible algorithm applies a key on the message, which aims to hide the message for sending or reveal it when it arrives at destination. Thus, it is evident that the success of such algorithm depends directly on the nature of the key used. This work presents a solution on this domain based on the use of metaheuristics for the design of unidimensional chaotic cellular automata (CA) used as a key generator for stream cryptographic algorithms.

Keywords: Stream Cipher Cryptography, Cellular Automata, Simulated Annealing, Tabu Search, Metaheuristics.

Agradecimentos

Aos meus pais pelo apoio incondicional e pelas palavras de incentivo nas horas mais difíceis. Aos meus avós pelos ensinamentos passados, que impactaram muito no meu crescimento pessoal e a todos os meus parentes, pelo encorajamento e apoio.

Ao professor Saulo e ao professor Stênio pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Ao amigos do Grupo de Educação Tutorial do curso de Ciência da Computação, pelo apoio e amizade ofertados durante todo o período em que convivemos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AC	Autômato Celular
ACO	*Ant Colony Optimization*
AES	*Advanced Encryption Standard*
AG	Algoritmo Genético
GRASP	*Greedy Randomized Adaptive Search Procedure*
PRNG	*Pseudorandom Number Generator*
SA	*Simulated Annealing*
SMS	*Short Message Service*

1 Introdução

1.1 Apresentação do tema e contextualização do problema

Na computação existe uma classe de algoritmos criptográficos denominados algoritmos de chave simétrica. Estes algoritmos usam chaves de criptografia fortemente relacionadas. Geralmente, em aplicações reais, uma única chave é escolhida para ser utilizada nos dois processos, criptografar e descriptografar a mensagem. Uma parte desses algoritmos de chave simétrica são chamados algoritmos criptográficos de fluxo ou simplesmente algoritmos de fluxo (Menezes et al, 1996).

Os algoritmos de fluxo são usados quando a mensagem não pode, por algum motivo, ser criptografada de uma só vez. Basicamente, a ideia por trás desses algoritmos é a de criptografar o que estiver disponível no momento. Eles atuam progressivamente sobre as partes da mensagem disponíveis e são úteis quando se faz necessário, por exemplo, enviar um fluxo contínuo de dados, como em uma rádio que precise transmitir seu sinal criptografado de forma ininterrupta.

A chave criptográfica usada nesses algoritmos tem que possuir algumas características que tornem a transmissão de dados segura. Ela deve preferencialmente ser variável no tempo e ser ao menos tão extensa quanto a mensagem enviada. A solução implementada nestes algoritmos para gerar uma chave com essas características é a utilização de uma espécie de gerador, que funciona como uma caixa preta e que, a cada instante de tempo, entrega um pedaço da chave para ser combinado com parte da mensagem. Esse gerador é acionado por uma chave menor, que serve para configurar os seus parâmetros e ativar uma chave de fluxo específica dentro dele. Criar um gerador eficiente é uma tarefa árdua e uma infinidade de métodos já foi estudada e implementada com esse propósito (Zeng et al, 1991). Um desses métodos consiste na utilização de estruturas discretas com comportamento evolutivo conhecidas como autômatos celulares.

Um autômato celular, em sua essência, é um modelo matemático capaz de simular os mais diversos fenômenos do espaço partindo de um conjunto de regras locais bem simples (Wolfram, 1982). Um autômato celular é formado por elementos menores que tem seu comportamento definido pelo comportamento dos seus elementos mais próximos e por uma regra que dita como será essa interação. No domínio criptográfico é usada uma classe especial de autômatos conhecidos como autômatos celulares caóticos (Wolfram, 2002). Autômatos caóticos apresentam comportamentos complexos difíceis de prever, o que faz deles ótimos mecanismos geradores para algoritmos criptográficos, uma vez que a chave gerada herda a imprevisibilidade e o caos do autômato que a gerou. Esse último apontamento faz com que seja interessante sempre buscar um conjunto de regras que atribua esta característica ao autômato.

A escolha deste conjunto de regras ideal é um problema de otimização combinatória que depende dos parâmetros usados na definição do autômato e do tipo de autômato usado, o que faz com que, na maioria dos casos, seja inviável computacionalmente explorar todo o espaço de soluções do problema.

Assim, faz-se necessária uma forma de encontrar uma solução satisfatória, mesmo que não seja a melhor possível, em um tempo razoável, justificando a escolha de métodos que direcionem esse processo de seleção de regras. Na literatura, existem alguns trabalhos destacando as vantagens e desvantagens no uso de alguns métodos com esta finalidade (Tomassini e Perrenou, 2001; Villela e Carvalho, 2007). Um ponto a ser verificado é se essas técnicas são capazes de obter soluções com qualidade suficiente e em tempo hábil para tornar o autômato celular, e a chave gerada por ele, núcleo criptográfico de uma aplicação prática.

1.2 Justificativa

A importância de sistemas que garantam a segurança de informações confidenciais é algo inquestionável nos dias atuais. Pode-se não perceber, mas grande parte dos equipamentos e sistemas possuem mecanismos que garantem, por exemplo, que seus dados estejam seguros durante uma transação bancária ou quando se está conversando com um parente distante por meio do seu telefone celular. O vazamento de uma simples informação privi-

legiada pode decidir o destino de uma guerra (Krischer e Weber, 2012) ou provocar uma série de complicações à ambientes como o corporativo (Canal Tech, 2014).

A utilização de autômatos celulares como mecanismos criptográficos já é algo bem sólido na literatura, sendo citado em diversos trabalhos correlacionados desde o trabalho precursor de Wolfram (1985). A facilidade de implementação e a versatilidade são só algumas das características que disseminam o uso desse tipo de abordagem.

Um autômato celular é capaz de criptografar uma quantidade significativa de bits que varia dependendo do tamanho do mesmo. Um autômato unidimensional de 16 bits, por exemplo, seria capaz de criptografar uma mensagem com cerca de 131.000 caracteres, o que equivale a aproximadamente 900 mensagens no Twitter ou 800 mensagens enviadas via SMS.

1.3 Objetivos

O objetivo desse trabalho é o desenvolvimento de abordagens baseadas em metaheurísticas para obtenção de autômatos celulares caóticos aplicados como método gerador em algoritmos criptográficos de fluxo, seguido da validação dos resultados obtidos durante o processo.

Como objetivos específicos, podem ser citados:

- Analisar o comportamento de algumas abordagens metaheuristicas frente ao problema apresentado;
- Discorrer sobre características inerentes ao problema que possam facilitar ou direcionar melhor o processo de obtenção de autômatos celulares caóticos;
- Validar a qualidade das soluções obtidas por meio de testes já consolidados na literatura.

1.4 Organização

O presente trabalho integra 6 capítulos. No primeiro capítulo é feita uma apresentação do problema, destacando a sua real importância no âmbito da segurança de sistemas e

por fim, é realizado um levantamento dos principais objetivos a serem alcançados.

No capítulo seguinte, Fundamentação Teórica, são apresentados os principais aspectos teóricos do problema analisado e o ferramental utilizado no decorrer do desenvolvimento deste trabalho. Neste capítulo são introduzidos conceitos importantes referentes à criptografia, mais precisamente à criptografia de fluxo, de autômatos celulares e das metaheurísticas analisadas, com o objetivo de familiarizar e preparar o leitor para as questões abordadas nos capítulos seguintes. O Capítulo 3 fornece um panorama geral do tema por meio de uma breve explanação de uma série de trabalhos presentes na literatura que se relacionam de alguma forma com este trabalho. No Capítulo 4 são apresentadas com detalhes as abordagens propostas para solucionar o problema apresentado.

O Capítulo 5 descreve os experimentos realizados e os resultados computacionais obtidos por meio das abordagens propostas, e é feita uma análise do comportamento das mesmas frente ao problema estudado e das soluções obtidas neste processo. No capítulo final são apresentadas as conclusões, além de possíveis linhas de estudo para trabalhos futuros.

2 Fundamentação Teórica

Nesse capítulo são apresentados os principais conceitos relacionados ao tema proposto. Conceitos referentes à criptografia na seção 2.1, à autômatos celulares na seção 2.2 e às metaheurísticas aplicadas na busca pela solução para o problema apresentado, na seção 2.4.

2.1 Criptografia

A criptografia compreende uma família de métodos responsáveis por codificar uma informação por meio da aplicação de uma transformação. Essa transformação tem como objetivo restringir o acesso a essas informações, escondendo o seu conteúdo original. O termo criptografia é de origem grega e é formado pela junção das expressões 'kryptos' e 'graphein', que significam respectivamente oculto e escrever. Esta técnica é tão antiga que existem registros de algo parecido com um sistema criptográfico de 4000 a.C no Egito (Singh, 2011). Hoje este tipo de técnica é amplamente utilizada e está presente em uma quantidade considerável dos sistemas e equipamentos presentes no dia a dia das pessoas (Wykes, 2016).

Os algoritmos criptográficos se dividem em dois grandes grupos (Menezes et al, 1996). No primeiro grupo se enquadram os algoritmos criptográficos assimétricos. Os algoritmos deste grupo possuem dois tipos diferentes de chave, uma chave pública e uma chave privada. A chave pública é usada para criptografar ou descriptografar uma mensagem e a chave privada é usada no processo inverso ao executado pela chave pública. Algoritmos de chave assimétrica têm toda sua segurança baseada no fato de que muitos são inspirados em problemas matemáticos que atualmente não admitem solução eficiente. Isso faz com que prever uma chave privada tendo como parâmetro sua chave pública seja uma tarefa inviável do ponto de vista computacional. O usuário só deve garantir que a chave privada permaneça em sigilo.

Um dos representantes mais conhecido deste grupo é o algoritmo RSA (Rivest,

Shamir e Adleman) (Rivest et al, 1978). O RSA tem seu funcionamento fundamentado na exponenciação modular de dois números inteiros primos grandes. A segurança deste algoritmo está relacionada ao fato de ainda não existir um algoritmo eficiente para fatorar números inteiros muito grandes. O RSA é uma solução tão robusta que ainda é amplamente utilizada como método criptográfico.

No segundo grupo estão os algoritmos criptográficos simétricos, que têm este nome graças à natureza das chaves usadas pelos mesmos. Estes algoritmos usam chaves criptográficas secretas fortemente relacionadas para o processo de criptografia e descriptografia. Na prática, uma única chave pode ser usada para os dois processos e na maioria dos casos reais é isso que ocorre. Dentro deste grupo os algoritmos são subdivididos em algoritmos criptográficos de bloco e algoritmos criptográficos de fluxo.

Algoritmos criptográficos de bloco operam sobre blocos de bits de um tamanho definido. Antes de passar pela funcionalidade principal do algoritmo de bloco, toda a mensagem tem que ser dividida em blocos de bits com o tamanho especificado. Caso a mensagem não tenha um número de bits múltiplo do tamanho do bloco especificado, então é necessário completar o último bloco com bits de um mesmo valor até alcançar o tamanho determinado.

2.1.1 Criptografia de Fluxo

Como já foi dito, os algoritmos de fluxo são parte de uma classe de algoritmos criptográficos denominados algoritmos de chave simétrica. Na criptografia de fluxo todas as operações são realizadas em pequenos fragmentos da mensagem (Ex.: Bits, bytes, caracteres). Uma forma simples de executar esse processo é chamada *one-time pad* (Miller, 1982), que consiste na combinação de cada bit da mensagem com um bit de uma sequência aleatória, isto por meio de uma operação matemática L. Devido a comutatividade e reversibilidade, a operação mais comumente usada neste processo é o ou-exclusivo (XOR). A comutatividade garante que a ordem dos operandos não interfira no resultado da operação, já a reversibilidade permite que uma operação seja desfeita, aplicando a mesma operação a um dos operandos e ao resultado da operação anterior. Com todas essas características, a operação XOR, por si só, pode ser considerada um método criptográfico razoável. A

operação de desencriptação pode ser realizada basicamente aplicando a operação inversa L_{inv} aos bits da mensagem cifrada e aos da sequência aleatória. No fim, todo este processo se resume à solução do sistema de equações lineares L descrito na equação 2.1. Onde x_i, y_i e k_i são, respectivamente, um bit da mensagem X, um bit cifrado e um bit da sequência aleatória K usada.

$$\begin{bmatrix} y_i = L(x_i, k_i) & i = 0, 1, .., |X| \\ x_i = L(y_i, k_i) & \end{bmatrix} \tag{2.1}$$

O algoritmo desenvolvido com base nesta técnica só pode ser considerado seguro se o sistema apresentado na equação 2.1 não admitir solução. Pode-se provar que este sistema não possui solução se duas condições forem obedecidas (Miller, 1982).

1. Para cada y_i, $x_i = 0$ e $x_i = 1$ são equiprováveis.

2. A sequência K usada for gerada por uma fonte de natureza aleatória.

A sequência aleatória usada em todo o processo é chamada chave de fluxo. Essa chave deve ter o mesmo número de bits da mensagem original e ser formada por bits que variam no tempo. Assim, cada bit da mensagem é criptografado por um bit diferente da chave. Um problema operacional é gerado ao utilizar-se uma chave de fluxo totalmente aleatória. Como a chave usada é a mesma para o processo de encriptação e desencriptação, como demonstrado na Figura 2.1, ela deve ser armazenada e repassada ao receptor, o que pode comprometer a segurança do algoritmo. Para contornar este problema, utiliza-se uma sequência de bits pseudoaleatória, que pode ser reconstruída caso necessário nos extremos da conexão e que não precise ser passada integralmente nesse processo.

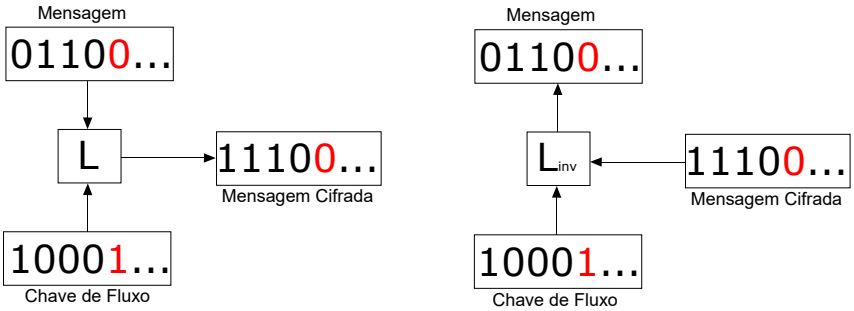

Figura 2.1: Processo de encriptação (esquerda) e desencriptação.

Alguma máquina ou algoritmo determinístico deve ser implementado em conjunto com o algoritmo de fluxo para gerar a chave pseudoaleatória. Esse elemento gerador deve ser capaz de reconstruir a chave quando necessário e garantir que ela tenha características aleatórias o suficiente para ser considerada segura. Esses algoritmos, em geral, têm um elemento disparador conhecido como chave semente, que é responsável por calibrar os parâmetros do gerador e ativar uma chave de fluxo específica dentro do mesmo. O emissor e o receptor, portanto, só precisam compartilhar a chave semente. Uma representação conceitual de como se dá o processo é mostrada na Figura 2.2.

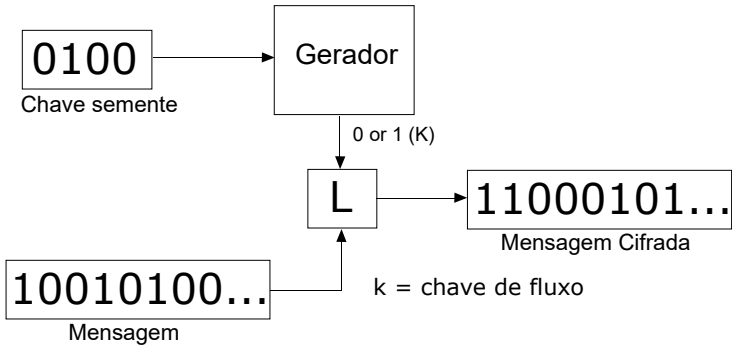

Figura 2.2: Processo de encriptação usando uma operação L e um gerador.

2.1.2 Mecanismo Gerador

Com base nos elementos apresentados na seção anteriores, fica evidente a importância de um mecanismo gerador bem construído no bom funcionamento de um algoritmo criptográfico de fluxo. Um gerador ideal é uma abstração para um algoritmo bem mais complexo, que têm seu principio de funcionamento fundamentando na geração de eventos que aparentam ser aleatórios. Segundo Bassham et al (2010), estes eventos são ditos aleatórios se forem gerados de forma independente uns dos outros e se nenhum evento posterior puder ser previsto, independente da quantidade de eventos já observados. Portanto, o gerador criado com base nestas ideias seria capaz de gerar uma sequência de elementos equiprováveis, o que é impraticável do ponto de vista computacional para uma quantidade suficiente de elementos a serem gerados (Bassham et al, 2010).

Neste sentido, no ambiente criptográfico propõe-se o uso de geradores que simulem fenômenos pseudo-aleatórios (PRNGs). Estes geradores usam funções determinísticas que são “ativadas” por um ou mais elementos chamados chaves sementes. Essas sementes

geralmente são criadas por meio de outros processos aleatórios e conferem a característica pseudo-aleatória do processo, uma vez que basta conhecer a semente e o algoritmo usados, para validar ou reproduzir qualquer sequência gerada. É importante ressaltar que nem todos geradores pseudo-aleatórios possuem aplicação criptográfica, já que os elementos em sua saída devem ser imprevisíveis e este fato não é observado em grande parte destes elementos (Paar e Pelzl, 2009). Um gerador com afinidade criptográfica pode ser modelado por meio de problemas matemáticos extremamente complicados, primitivas criptográficas (cifras, *hashs*) ou usando outras técnicas como algoritmos evolutivos (Poorghanad et al, 2008) e autômatos celulares (Wolfram, 1986).

2.2 Autômatos Celulares

Em 1940, Stanislaw Ulam e John von Neumann criaram um modelo matemático conhecido como autômato celular (Wolfram, 1982). Mais tarde, Wolfram (1984) introduziu este mesmo modelo no estudo do comportamento de alguns sistemas extremamente complexos. Neste artigo o autor define um autômato celular como um sistema matemático construído por meio de uma série de componentes idênticos simples que juntos são capazes de apresentar comportamentos complexos.

Na computação estes componentes são denominados células e estão dispostos em uma estrutura n-dimensional governada por uma série de regras, que definem o comportamento do sistema como um todo. Cada célula possui um valor atribuído que representa um dentre os vários estados possíveis levantados na formulação de um problema. Ligado ou desligado, frio ou quente, 0 ou 1 são só alguns exemplos de estados que podem ser usados em um autômato celular. Um estado associado a uma célula é obtido por meio da aplicação de uma regra estática, que depende do estado dessa célula e do estado de outras células localizadas em suas proximidades (vizinhança) em um instante de tempo anterior. O conceito de vizinhança é variável e depende da forma usada para definir a proximidade entre as células, porém uma ideia bastante intuitiva é a de delimitar um raio na qual todas células inseridas são ditas vizinhas como visto na Figura 2.3.

Villela e Carvalho (2007) definem uma regra como uma espécie de máquina de estados finitos que mapeia para cada vizinhança um estado correspondente como pode

ser visto na Figura 2.4. Um AC pode ter uma regra definida para cada célula ou uma regra geral aplicada a todas, e os autômatos resultantes são caracterizados, respectivamente, como heterogêneos e homogêneos (Villela e Carvalho, 2007). Partindo de uma configuração inicial, a aplicação sucessiva do conjunto de regras causa no AC uma propagação de efeitos que, confere o caráter evolutivo e gera as configurações passíveis de serem assumidas por ele como observado na Figura 2.5.

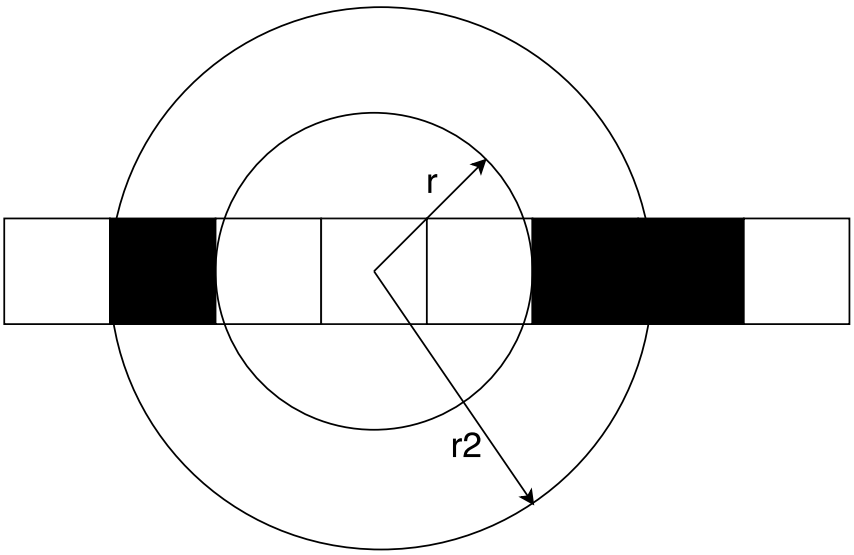

Figura 2.3: Vizinhança de um autômato celular definida a partir de dois raios diferentes.

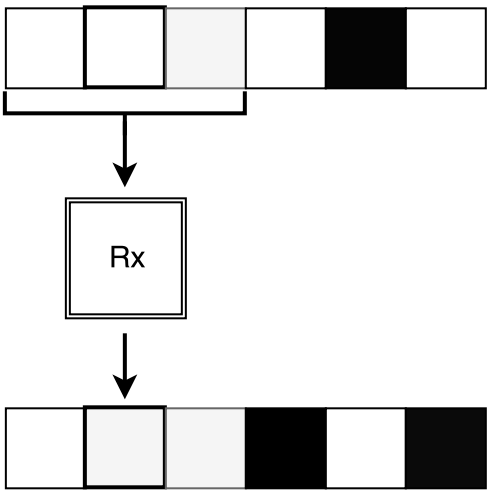

Figura 2.4: Exemplo da aplicação de uma regra Rx sobre uma célula de um AC.

Os autômatos podem ainda ser classificados em quatro grandes grupos, como sugere o trabalho desenvolvido em Wolfram (2002), que concluiu a partir de uma análise empírica que, apesar das diferentes regras e configurações iniciais, os AC's depois de um

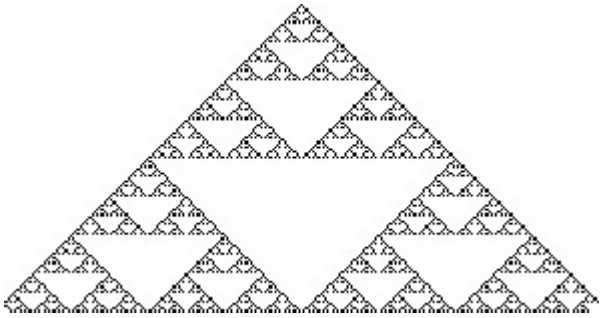

Figura 2.5: Algumas gerações de um AC unidimensional

número suficiente de gerações apresentavam um dos seguintes comportamentos:

- Classe I: A partir de um estado inicial qualquer, todas as células convergem para um mesmo estado terminal.
- Classe II: Os AC's dessa classe geram estruturas estáveis ou periódicas muito simples.
- Classe III: Os AC's dessa classe geram estruturas com comportamento caótico por um número suficiente de gerações.
- Classe IV: Os AC's dessa classe possuem um comportamento muito mais complexo, gerando estruturas espaço-temporais que em raros casos podem ser persistentes.

Esta classificação está ilustrada na Figura 2.6.

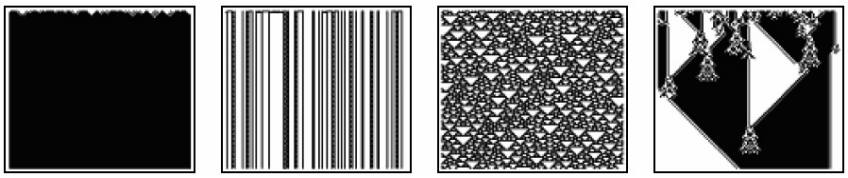

Figura 2.6: Classes observadas por Wolfram: (I) - (IV)

Em um primeiro momento pode parecer que AC's são aplicáveis a uma conjuntura de problemas bem específicos, porém isso não é verdade, já que estranhamente não é difícil encontrar na natureza, sistemas complexos com o comportamento perfeitamente reproduzível por AC's. Esta característica está na essência da complexidade inerente a estes sistemas, que está na integração e interação entre várias partes simples com algum propósito definido (Wolfram, 1984). Wolfram (2002) sintetizou essas observações com o

objetivo de ratificar a contribuição e as descobertas baseadas na utilização de autômatos celulares nas mais diversas areas do conhecimento.

Souza (2001) fornece uma descrição de alguns fenômenos naturais e sociais com estas características e apresenta um modelo desenvolvido com base em AC's para simular cada um destes fenômenos. Sistemas mecânicos, reações químicas, processos de formações de cristais, modelos de mobilidade urbana, propagação de epidemias são alguns dos exemplos de fenômenos apresentados pelo autor. Em um desses exemplos, o autor fornece a descrição de um modelo usado para reproduzir o comportamento de um sistema do tipo predador/presa em um ecossistema natural. Neste modelo, cada célula do AC caracteriza uma parte do ambiente a ser representado e é classificada de acordo com os elementos dominantes (vegetação/animal). Predador e presa se movimentam aleatoriamente sobre o ambiente representado pelas células do AC e quando o predador depara-se com uma presa, ele come a presa e se move para a célula onde ela se encontrava. Um tempo de procriação é associado para predador e presa e um tempo de vida é associado ao predador. A ideia é que o modelo dê um indicativo de como se dará a expansão das espécies e suas interações dentro do ambiente analisado.

Zuse (1969) foi além e defendeu o conceito de que todo o universo era regido por leis físicas essencialmente discretas, o que em teoria significa dizer que existe uma máquina capaz de computar a evolução do universo expandindo todas as possibilidades durante este processo. Esta máquina funcionaria como um único autômato celular com todo o universo em sua saída.

2.3 Entropia

Na termodinâmica, a entropia é uma grandeza associada à desordem de um sistema físico. Uma variação de entropia no sistema ocorre em situações onde há transferência de energia na forma de calor. Isto acontece porque a energia transferida nunca é totalmente transformada em trabalho. Assim, uma parte dessa energia dissipada é transformada em entropia, aumentando a desordem do sistema (Callen, 1985).

Na teoria da informação, a entropia está relacionada à quantidade de informação necessária para compreender um dado fenômeno (Shannon, 1948). Quando não há como

prever o comportamento do fenômeno, a análise é feita de forma probabilística, considerando a probabilidade associada p_i à ocorrência de cada evento E_i. A entropia de informação é a quantidade média de informação por evento (Pavão, 2011) e é representada por:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i, \tag{2.2}$$

O máximo da equação 2.2 representa um fenômeno com um certo número de variáveis (eventos) de difícil previsão. Pode-se afirmar que a entropia de informação de um determinado fenômeno está diretamente relacionada à quantidade de eventos e à dificuldade de prever esses eventos.

De forma sucinta, a entropia da informação pode ser descrita como uma forma de prever a quantidade de informação que se têm sobre um determinado assunto. Quanto mais próximo do valor máximo de Entropia mais incerto e imprevisível (Shannon, 1948) é o fenômeno analisado. Estas características fazem dessa grandeza um bom parâmetro para medir a qualidade de chaves geradas por sistemas criptográficos.

2.4 Metaheurísticas

Existem problemas na computação tão complexos que não se conhece um algoritmo eficiente para fornecer soluções exatas para os mesmos em tempo polinomial. Com isso, é justificável a preocupação em encontrar e aprimorar métodos que forneçam soluções aproximadas para estes problemas e que possam ser aplicadas em substituição a soluções exatas em condições controladas. As metaheurísticas são exemplos de métodos desenvolvidos com esta finalidade (Goldbarg et al, 2015).

Osman e Laporte (1996) definem uma metaheurística como um processo iterativo responsável por guiar heurísticas subordinadas a ele, e que combina diversas técnicas com o intuito de explorar o espaço de busca de um problema de forma eficiente.

Um outra definição é fornecida em Dorigo e Stützle (2004). Neste trabalho, os autores afirmam que uma metaheurística pode ser entendida como um método heurístico mais geral criado para direcionar uma heurística mais específica na solução de um problema.

Goldbarg et al (2015) apresentam duas formas diferentes de classificação para metaheurísticas. Na primeira, o critério usado na classificação tem relação com a estratégia usada para obter as soluções. Já na segunda, o critério usado tem relação com o tipo de estrutura de vizinhança empregado na mesma. Uma estrutura de vizinhança define uma forma de obter soluções vizinhas a uma dada solução analisada.

- Segundo a estratégia usada na obtenção das soluções:
  - Construtivas: Esta classe compreende as metaheurísticas que tem soluções organizadas passo a passo de modo crescente e que podem ou não ser melhoradas em alguma fase posterior do algoritmo. O ACO (Dorigo e Stützle, 2004) é um exemplo de metaheurística desta classe.
  - Evolutivas: As soluções obtidas são construídas por meio de transformações em um conjunto de soluções já conhecidas. Nesta classe existem métodos responsáveis por determinar dentre as soluções existentes quais serão reconhecidas em iterações posteriores do algoritmo.
  - de Decomposição: As soluções são construídas a partir da solução de subproblemas mais fáceis de serem resolvidos.
  - de Informação Compartilhada: As soluções são organizadas a partir de vários agentes que constroem as soluções por meio do auxílio de algum método responsável por disseminar alguma informação entre os agentes participantes que guie os neste processo.
- Segundo o tipo de estrutura de vizinhança utilizado:
  - de Vizinhança Fixa: Metaheurísticas desse tipo tem suas estruturas de vizinhança definidas no ínicio de sua execução que permanecem imutáveis durante todo o processo. O GRASP (Feo e Resende, 1989) e o *Simulated Annealing* (Metropolis et al, 1953) são metaheurísticas que fazem parte desta classe.
  - de Vizinhança Flexível: Nestes algoritmos as estruturas de vizinhança podem ser flexibilizadas em determinadas situações especiais, porém este processo geralmente é feito de forma pouco sistemática. A Busca Tabu (Glover, 1986) é um exemplo de metaheurística desta classe.

– de Vizinhança Variável: Metaheurísticas dessa natureza adotam a possibilidade de exploração de suas diversas estruturas de vizinhança de forma sistemática. O VNS (Mladenović e Hansen, 1997) é um exemplo de metaheurística desta classe.

Alguns dos problemas mais emblemáticos da classe NP completo já foram extensivamente explorados usando metaheurísticas dos mais diversos tipos. Um destes problemas é o PCV (Problema do Caixeiro Viajante), que consiste em determinar a menor rota para visitar uma série de pontos sem repetir nenhum (Lenstra, 2009) que corresponde ao ciclo hamiltoniano de custo mínimo em um grafo com esses pontos. Este problema tem, por exemplo, abordagens construídas baseadas em *Simulated Annealing* (Dalle e Martins, 2004), AGs (Brady, 1985), ACO (Stützle e Dorigo, 1999), entre outros.

Nesse trabalho são analisadas abordagens desenvolvidas com base nas metaheurísticas *Simulated Annealing*, Busca Tabu e ACO, usando a entropia como função objetivo. Os detalhes destas metaheurísticas são apresentados nas subseções seguintes.

2.4.1 Busca Tabu

A Busca Tabu é um método de exploração de vizinhança proposto inicialmente por Fred Glover (Glover, 1986). Neste trabalho o autor define-a como uma metaheurística combinada à outra heurística. A ideia central da Busca Tabu é avaliar a vizinhança imediata de uma solução em potencial, com o objetivo de atuar em problemas com ótimos locais ou em regiões muito homogêneas do espaço de soluções.

O algoritmo de Busca Tabu representado pelo pseudocódigo do Algoritmo 1, funciona de forma muito simples. A cada iteração, o algoritmo explora a vizinhança da solução corrente s e seleciona o seu vizinho mais promissor s', segundo alguma métrica definida. Durante este processo, soluções eventualmente já visitadas são evitadas, visto que, em princípio, uma busca eficiente não deveria visitar partes do espaço de soluções já analisadas (Goldbarg et al, 2015). Para manter este controle, Fred Glover conferiu à sua metaheurística uma espécie de memória, que na prática é representada por uma lista restritiva T com um histórico das soluções visitadas pelo algoritmo. Isso evita eventuais ciclos e que o algoritmo retorne a um ótimo local previamente visitado, melhor explorando

o espaço de soluções. O algoritmo é executado até que uma condição de parada seja atingida, que pode ser um número máximo de iterações sem melhora na solução ou até que a melhor solução chegue a um limite definido.

O problema é que, manter um histórico com todas soluções é uma tarefa do ponto de vista computacional muito árdua (Goldbarg et al, 2015). Uma forma de contornar esta situação é manter um registro só de alterações nas soluções ou em variáveis. Estas alterações são caracterizadas como movimentos e vão de simples alterações de bits em uma configuração binária até atribuições mais complexas, como permutações em vértices de um grafo. A lista restritiva conhecida como lista tabu então passa a ter os últimos movimentos executados pelo procedimento e não as soluções. O termo tabu remete a proibição e ilustra a condição imposta aos movimentos armazenados na lista. O propósito desta lista não é o de impedir a repetição de um movimento e sim evitar que um outro movimento seja desfeito (Glover, 1986).

Entretanto, o caráter proibitivo desta abordagem pode ser um entrave, quando se faz necessário revisitar uma parte do espaço de soluções. Dessa forma, fez-se necessário uma forma de, em condições especiais, ignorar o caráter tabu de um movimento. Glover (1986) pensou em uma função especial (função de aspiração), utilizada no sentido de permitir que um movimento tabu seja realizado, por exemplo, se houver uma melhora no valor da função objetivo ou se todos os movimentos disponíveis forem tabus.

2.4.2 *Simulated Annealing*

A técnica de *Simulated Annealing* (SA) é uma metaheurística baseada em um processo termodinâmico chamado recozimento. Este processo consiste em submeter um material a altas temperaturas e depois controlar seu resfriamento levando-o a um estado de energia mínima. Neste estado, o material apresenta um grau de perfeição estrutural superior ao que apresentava anteriormente (CIMM, 2016).

Kirkpatrick et al (1983) percebeu que poderia simular esse processo e aplicar a problemas de otimização combinatória, dando origem ao Simulated Annealing. O SA usa o conceito de energia associado a cada solução. Alterações em uma solução podem levar a um aumento ou diminuição dessa energia. O objetivo é achar a melhor solução através

Algoritmo 1: Busca Tabu (Goldbarg et al, 2015)

Entrada: solução inicial s_0;
nº máximo de iterações $IterMax$;
nº máximo de iterações sem melhora em s^* $IterMaxMelhora$;
Saída: solução s^*;
Data: Sendo s'^* melhor solução possível na vizinhança $\eta(s)$ da solução s, L é a lista Tabu, $p(s'^*)$ é o movimento feito em s para gerar s'^* e $A(f(s))$ é a função de aspiração utilizada.

```
1  início
2      Iter ← 0;                                  // Contador do nº de iterações
3      IterSMelhora ← 0;  // Contador do nº de iterações sem melhora
4      s ← s_0, s* ← s;
5      L ← ∅;
6      enquanto Iter ≤ IterMax e IterSMelhora ≤ IterMaxMelhora faça
7          Iter ← Iter + 1;
8          IterSMelhora ← IterSMelhora + 1;
9          s = s'* ∈ η(s) ∧ p(s'*) ⊄ L ∨ f(s'*) satisfaz A(f(s'*));
10         T = T ∪ p(s'*);                        // Atualiza lista Tabu
11         se f(s) > f(s* então
12             s* ← s;
13             IterSMelhora ← 0;
14         fim se
15     fim-enquanto
16 fim
```

da maximização de uma função associada a essa energia.

O algoritmo do SA, retratado no algoritmo 2, inicia com uma solução s_0 qualquer e, a cada iteração, a solução corrente s é modificada aleatoriamente, provocando uma variação na energia do sistema Δf. Alterações que levem a um aumento na energia do sistema são aceitas de imediato. Um decréscimo é aceito segundo uma probabilidade $p = e^{\frac{-\Delta f}{KT}}$, onde Δf é a diferença de energia associada e T é a temperatura atual. O valor K é um valor conhecido como constante de Boltzmann (Villela e Carvalho, 2007). Inicialmente, a temperatura T assume um valor alto, T_0 e decai ao longo da execução. Para cada temperatura são geradas soluções por um número definido de iterações. O algoritmo deve convergir para o equilíbrio térmico, situação onde a média do valor da função objetivo estaciona perto do valor já conhecido. Neste ponto a temperatura sofre um decréscimo segundo a equação 2.3.

$$T_n = \alpha\, T_{n-1}, \tag{2.3}$$

onde T_n é a temperatura em um estágio de resfriamento n e α é um fator associado à estratégia de resfriamento adotada.

O procedimento é finalizado quando a temperatura chega a um valor próximo de zero, que indica um estado de "cristalização" do sistema. Neste estado, qualquer alteração realizada não provocará um aumento na função objetivo e nenhuma solução com o valor pior que a melhor solução pode ser aceita. Espera-se que nesta situação a melhor solução obtida esteja próxima a um ótimo global.

Algoritmo 2: *Simulated Annealing*

Entrada: solução inicial s_0;
temperatura miníma T_{min};
temperatura inicial T_0;
fator de decréscimo na temperatura α;
Saída: solução s^*;

```
1  início
2      T ← T_0;
3      s ← s_0;
4      enquanto T ≥ T_min faça
5          enquanto enquanto não houver equilíbrio térmico na temperatura T
           faça
6              selecionar uma solução s' dentro da vizinhança de s;
7              Δf ← f(s') − f(s);
8              se Δf ≥ 0 então
9                  s ← s';
10             senão
11                 s ← s' com uma probabilidade p = e^(−Δf/KT);
12             fim se
13         fim-enquanto
14         T ← α(T) ;
15     fim-enquanto
16 fim
```

2.4.3 *Ant Colony Optimization*

O algoritmo ACO (*Ant Colony Optimization*) é uma metaheurística populacional evolutiva apresentada por Dorigo e Di Caro (1999), e que desde então já serviu como base para o desenvolvimento de algoritmos para uma série de problemas (Dorigo e Stützle, 2009). Ela funciona basicamente reproduzindo o comportamento observado em uma colônia de formigas. As formigas geralmente tendem a encontrar o menor caminho entre a colônia e seu alimento (Dorigo e Stützle, 2009). Durante esse percurso elas vão depositando uma

substância chamada feromônio, que marca esse trajeto, fazendo com que outras formigas tenham uma tendência a repetir o caminho marcado. Essa tendência depende da quantidade de feromônio depositada na trilha, sendo o caminho com a maior concentração de feromônio o mais provável de ser escolhido.

Inicialmente, quando não há feromônio sobre nenhum caminho, as formigas saem aleatoriamente a procura de alimento e as que obtêm sucesso retornam gerando os primeiros caminhos marcados. As outras formigas então passam a ter uma chance maior de escolher esses caminhos reforçando os seus feromônios. Com o tempo, os melhores caminhos vão recebendo as maiores quantidades de feromônios, sendo esses caminhos os mais escolhidos pelas formigas. Os feromônios podem evaporar e dessa forma os caminhos menos visitados perdem feromônios fazendo com que as outras formigas passem a evitá-los. Essa evaporação é mais lenta em caminhos com uma grande concentração de feromônios.

O ACO tenta se apoderar destes conceitos criando uma espécie de colônia virtual, onde as formigas exploram o espaço de soluções de um problema a procura das melhores fontes de alimentos, que correspondem as melhores soluções geradas. Geralmente esse espaço de soluções é modelado em um grafo e as formigas depositam e evaporam os feromônios nas arestas e vértices desse grafo. Uma versão em pseudocódigo do ACO está descrita no Algoritmo 3.

Algoritmo 3: *Ant Colony Optimization*

Entrada: soluções iniciais S_0;
número máximo de iterações $IterMax$;
Saída: solução s^*;

```
1  início
2      S ← S_0;                                  // Primeira geração de formigas
3      Iter ← 0;                                 // Contador do número de iterações
4      enquanto Iter ≤ IterMax faça
5          para formiga k = 1 até N faça
6              construir a solução para a formiga k;
7              se f(s^k) > f(s^*) então
8                  s^* = s^k;
9              fim se
10         fim-para
11         atualizar os feromônios;   // Depositar e evaporar feromônios
           nas arestas
12         Iter ← Iter + 1;
13     fim-enquanto
14 fim
```

3 Revisão da Literatura

3.1 Trabalhos Relacionados

Em 1985, Wolfram (1985) abriu um novo horizonte no campo criptográfico, ao definir as diretrizes iniciais do uso de autômatos celulares como mecanismos fundamentais de dispositivos criptográficos de fluxo. O autômato celular usado por ele possuía uma única dimensão com dois estados possíveis associados a cada célula, 0 ou 1. A atualização de cada célula a_i era feita de forma sincronizada segundo a equação 3.1.

$$a_i \leftarrow a_{i-1} \text{ XOR } (a_i \text{ OR } a_{i+1}) \tag{3.1}$$

O estado inicial do AC era usado como um elemento disparador (chave), e o restante das configurações geradas com o decorrer do tempo, conferia o caráter randômico ao sistema. O texto criptografado C podia ser obtido combinando a informação armazenada em sua forma binária P com as células a do AC segundo a equação 3.2. O processo inverso funciona da mesma forma, basta alterar a mensagem pelo texto cifrado.

$$C_i = P_i \text{ XOR } a_i \tag{3.2}$$

O autômato celular usado por Wolfram (1985) é um tipo especial de AC conhecido como AC caótico. AC's caóticos fazem parte da classe 3 de Wolfram e são capazes de gerar padrões complexos mesmo com configurações iniciais bem simples como na Figura 2.6. Uma evolução desse tipo de AC é considerada uma computação bem complexa, mais complexa e sofisticada que muitos sistemas computacionais. A maior parte dos AC's caóticos é não-linear e isso faz com que criar um método geral de predição seja muito difícil (Wolfram, 1986). Este método talvez nem exista, já que Wolfram (1986) desconfiava que esse era um problema computacionalmente irredutível.

Um AC caótico incorpora todo o caos e imprevisibilidade almejados na construção

de sistemas criptográficos eficientes, tornando-o, como já foi comentado anteriormente, um ótimo mecanismo gerador para esses sistemas.

Partindo desta constatação, o autor faz uma análise sobre a segurança de sistemas criptográficos criados com base neste tipo de técnica e chega a conclusão de que ela está principalmente na dificuldade de em um dado instante de tempo, descobrir a configuração atual do AC. Não é conhecido nenhum algoritmo eficiente para realizar esta tarefa em tempo polinomial. Além disso, é praticamente impossível fazer alguma análise estatística em sequências menores que um ciclo do AC, dada a complexidade das interações entre os elementos com o decorrer das gerações.

Depois deste trabalho de Wolfram (1985), uma série de abordagens foram desenvolvidas usando AC's aplicados ao ramo criptográfico.

Em Tomassini e Perrenou (2001), por exemplo, é proposta uma abordagem evolucionista baseada na combinação entre Algoritmos Genéticos (Goldbarg et al, 2015) e AC's. Os autores defendem que uma simples combinação entre um conjunto de regras bem específico aplicada ao AC utilizado já garantiria a segurança dos dados criptografados. Ele obteve essas conclusões ao observar que este conjunto de regras sempre aparecia nos indivíduos mais aptos gerados por seu algoritmo genético. O estudo dos autores se estendeu a autômatos de 1 e 2 dimensões mostrando as vantagens e as desvantagens no uso de cada um desses tipos de autômatos. Os testes realizados englobavam cenários onde o sistema proposto estava implementado em hardware e em software, com o objetivo de estudar a qualidade da solução desenvolvida em situações mais próximas da realidade.

Em 2004, Seredynski e Bouvry (2004) propuseram um conceito de um sistema de encriptação baseado na utilização de autômatos celulares unidimensionais reversíveis aplicados a criptografia de blocos. O sistema funcionou com desempenho bem semelhante a um modo bem conhecido de encriptação chamado CBC(Cypher Block Chainning). Este modo foi inventado em 1976 e funciona da seguinte forma: Cada bloco é combinado por meio de uma operação XOR com o bloco criptografado anteriormente (Singh, 2011). Com isso, é garantido que cada bloco criptografado terá alguma dependência com os blocos da mensagem em texto puro processados até o momento. No final do artigo, os autores concluem que esse tipo de abordagem baseado no uso de um único AC reversível não é

capaz de assegurar a segurança de um sistema deste tipo e oferecem um possível caminho por meio da utilização de múltiplos AC's com o mesmo propósito ao apresentado neste trabalho.

Em Villela e Carvalho (2007), os autores analisaram extensivamente os problemas de se utilizar apenas as combinações das regras sugeridas por Tomassini e Perrenou (2001) e propuseram uma abordagem que utiliza a técnica de *Simulated Annealing*, onde foca-se na geração de chaves de fluxo baseadas em autômatos celulares caóticos unidimensionais de 8 bits. Os autômatos usados são heterogêneos, ou seja, cada célula tem uma regra que pode ou não ser idêntica a outras e que define como será o seu comportamento com o decorrer das gerações. Todas as células estão submetidas a uma vizinhança de tamanho 3. Em essência, todas essas características foram usadas com o objetivo de possibilitar que a solução gerada se tornasse ainda mais aleatória. Os autores usaram como função objetivo do *Simulated Annealing*, a função de Entropia apresentada com detalhes na seção 2.3. Em seus testes, os autores conseguiram validar a abordagem desenvolvida, gerando soluções consideradas ótimas em um tempo médio de 1 hora. Com esses testes os autores conseguiram ainda, extrair uma característica importante deste problema, o fato da posição em que uma regra aparece interferir de forma direta na qualidade dessa regra, adicionando mais complexidade ao processo como um todo. Por fim, os autores enfatizam a dificuldade de criar um método capaz de gerar AC's caóticos dada a natureza extremamente combinatória do problema. O problema da abordagem desenvolvida por Villela e Carvalho (2007) está no fato de que autômatos de 8 bits geralmente são insuficientes para a maioria dos problemas reais, já que só podem gerar 256 combinações diferentes.

Uma outra proposta bastante interessante é a desenvolvida por Diwakar et al (2013). Neste trabalho os autores apresentam um sistema criptográfico baseado em dois níveis de segurança. A ideia é que esta configuração permita que os dados continuem sendo trafegados de forma segura mesmo que haja uma violação em um dos níveis, desde que o outro esteja funcionando corretamente.

O primeiro nível é fundamentado sob um ramo da criptografia conhecido como Esteganografia (Raggo e Hosmer, 2013), que estuda uma série de técnicas que provem formas de esconder uma informação em outra informação. Com isso é possível, por exemplo,

esconder a transmissão de dados altamente sigilosos em transmissões de informações que em um primeiro momento são consideradas comuns. Neste primeiro nível, a informação a ser transmitida é combinada com uma imagem que vai parecer ser a informação principal a ser protegida. Esta combinação é feita por um método conhecido como Substituição do Bit Menos Significativo (LSB), que consiste em utilizar os bits menos significativos de cada pixel da imagem para transmitir os fragmentos da informação confidencial(Raggo e Hosmer, 2013). Segundo os autores, em uma imagem com 24 bits de profundidade de cor, é possível esconder 3 bits de informação em cada pixel da imagem sem que o olho humano perceba que a imagem foi alterada. Isto é possível porque para cada pixel da imagem são destinados 3 bytes, um byte para cada canal de cor correspondente no sistema RGB (Red Green Blue) e uma troca no bit menos significativa em cada canal não produz quase nenhuma mudança significativa na informação aparente (Gonzalez e Woods, 2000). Essa etapa termina com a atribuição de uma chave definida pelo emissor para a imagem resultante do processo de combinação, de forma a garantir que somente consiga ter acesso a imagem quem possuir a chave atribuída a mesma.

No segundo nível a imagem passa por uma criptografia baseada no uso de autômatos celulares bidimensionais reversíveis. A chave de segurança atrelada a imagem também passa pelo mesmo processo. Os AC's usados foram definidos por meio da vizinhança de Moore que pode ser compreendida por meio da figura 3.1. Todo o processo é direcionado pelas regras 2, 8, 32 e 128. Num primeiro momento um bloco de 64 bits da imagem é alterado por um AC definido pela regra 2 e logo depois esse mesmo bloco é alterado por um AC definido pela regra 8, paralelamente a chave passa por esse mesmo procedimento. Numa segundo momento, chave e bloco de bits da imagem são unidos e passam por duas alterações sucessivas, uma por meio de um AC de regra 32 e outra por um AC de regra 128, respectivamente na ordem que ocorre o processo. Depois desse passo, a informação está pronta para ser transmitida. Para descriptografar a imagem basta seguir o caminho inverso do processo, já que, por definição, são usados AC's reversíveis durante todo o procedimento. No final do artigo, os autores apresentam um exemplo bem simples da aplicação do seu algoritmo com o objetivo de demonstrar na prática como o processo ocorre.

Com algumas alterações é possível adaptar a abordagem para funcionar como um algoritmo criptográfico de fluxo, porém essas adaptações podem gerar problemas na velocidade de transmissão dos dados, uma vez que nem sempre é possível ter um bloco disponível para criptografar, fazendo com que seja necessário reter a informação disponível até que seja possível utilizar o algoritmo. Um outro problema desse tipo de abordagem está no pressuposto pelos autores de que esses dois níveis de segurança implementados já seriam capazes de garantir uma transmissão segura dos dados sem testes específicos sem um estudo mais substancial de desempenho e viabilidade do modelo desenvolvido.

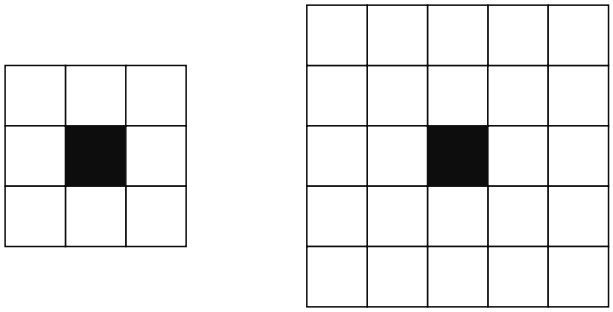

Figura 3.1: Vizinhança de Moore de raios 1 e 2.

Em 2014, Bouchkaren e Lazaar (2014) propuseram uma solução baseada na utilização de autômatos celulares bidimensionais reversíveis submetidos a vizinhança de MARGOLUS. Na vizinhança de MARGOLUS, o AC é dividido em blocos de um tamanho definido e submetidos a um particionamento que varia com o intervalo de tempo analisado. O procedimento adotado pelo algoritmo a cada intervalo de tempo fica restrito aos limites de cada bloco. Este procedimento pode ser encontrado com detalhes em (Tyler, 2016). O autor escolheu esta vizinhança porque a mesma permite a criação de AC's reversíveis. A reversibilidade de um AC está relacionada com a capacidade do mesmo em retornar a sua configuração depois de evoluir por um número qualquer de gerações. Essa reversibilidade garante mais segurança ao processo de criptografia.

No algoritmo desenvolvido por Bouchkaren e Lazaar (2014), a mensagem é dividida em blocos de tamanho fixo e passa por um processo parecido com o que ocorre na vizinhança de MARGOLUS. Para fins de comparação, o autor construiu o sistema proposto e simulou uma entrada, analisando o comportamento da sua abordagem frente ao algoritmo AES (Daemen e Rijmen, 2002). O problema desse tipo de abordagem aplicado

em um domínio como a criptografia de fluxo, está na necessidade de ter de antemão um bloco de tamanho fixo para começar o processo. Este fato pode provocar um atraso na transmissão, já que um bloco desse tamanho pode não estar disponível, fazendo com que o algoritmo tenha que reter informação até que o bloco esteja disponível.

4 Abordagens Propostas

Nesse trabalho foram desenvolvidas abordagens baseadas nas metaheurísticas *Simulated Annealing*, Busca Tabu para otimizar a escolha de autômatos celulares, para que sejam utilizados como mecanismos geradores de chave de fluxo (Brito et al, 2017). Uma versão baseada na metaheurística populacional ACO também foi implementada, porém essa versão não se mostrou competitiva se comparada às outras abordagens desenvolvidas. Inicialmente, são apresentadas com um maior detalhamento questões referentes ao problema e sua modelagem, e numa fase posterior uma descrição de todas as abordagens construídas.

4.1 O Problema

Os trabalhos de Wolfram (1985) e Wolfram (2002) já fornecem uma ideia do quão difícil pode ser o processo de geração de autômatos celulares com potêncial para serem aplicados em sistemas criptográficos reais. De fato, Villela e Carvalho (2007) já destacavam a natureza altamente combinatória deste tipo de problema e relacionavam esta característica a eficiência deste tipo de abordagem como mecanismo aplicado em sistemas criptográficos.

Wolfram (2002) atribui estas características a um tipo específico de AC na classificação proposta por ele. O autor define um AC da classe 3 ou caótico como um AC capaz de gerar padrões complexos difíceis de prever por um número suficiente de gerações. O processo de geração desses AC's, com base em uma configuração inicial, depende da definição de um conjunto de regras que estimule o surgimento deste comportamento caótico em suas gerações. Essa tarefa é bastante árdua do ponto de vista computacional devido ao seguintes fatores: As regras são fortemente relacionados umas com as outras e a posição com que as regras aparecem no AC também é altamente relevante neste processo. O primeiro fator está altamente ligado ao caráter evolutivo da abordagem e a influência local do conjunto de regras sobre a vizinhança de uma célula. A Figura 4.1 dá uma ideia do efeito em cascata observado na aplicação sucessiva do conjunto de regras sobre o autômato

e da relação temporal que existe entre regras que aparentemente não se relacionam no autômato. Essa relação está representada pela sobreposição entre as pirâmides nos instantes de tempo $t = 3$ e $t = 4$. A segunda constatação, observada por Villela e Carvalho (2007) em seus estudos, aumenta ainda mais a complexidade do problema, visto que este fator força o método empregado na solução do problema a encontrar permutações bem específicas dentro do conjunto de regras associadas a um AC, que geralmente estão em um número bem reduzido frente ao universo possível (Villela e Carvalho, 2007).

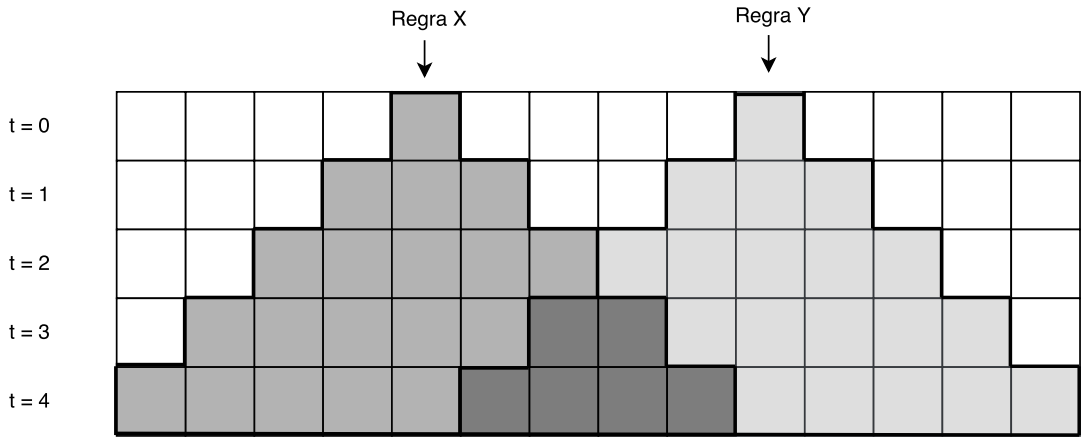

Figura 4.1: Influência das regras X e Y sobre as células de um AC e relação temporal entre elas

Dessa forma, justifica-se o uso de metaheurísticas no processo de obtenção destes conjunto de regras que conferem estas características caóticas a um AC. A função a ser otimizada pelas metaheurísticas usadas neste trabalho é a Entropia descrita na seção 2.3. A Entropia funciona como indicativo do pertencimento de um AC com um conjunto de regras específico a classe 3 de Wolfram, conforme já foi explicitado na mesma seção onde a função foi descrita e nos capítulos iniciais. Com base nestas ideias, pode-se modelar o problema da seguinte forma:

$$\begin{gathered}\text{Seja um autômato celular M} = (\Gamma, Q, \omega, N, \delta), \text{ definir } \delta \text{ tal que:} \\ \max H(M^*) \\ 0 \leq H(M^*) \leq |M|\end{gathered} \tag{4.1}$$

onde M^* representa todas as configurações possíveis (gerações) do autômato M, H representa a função de Entropia, Γ representa uma estrutura n-dimensional usada na modela-

gem do AC, Q é o conjunto de estados possíveis, N representa a vizinhança utilizada e $\delta = \{r_0, r_1, ..., r_{|M|-1}\}$ define o conjunto de regras associados a este autômato. Uma regra $r_i \in \delta$ define o estado futuro da célula i autômato da seguinte forma $r_i : Q^k \rightarrow Q$. Os parâmetros ω e $|M|$ representam, respectivamente, a configuração inicial do AC e o seu tamanho.

4.2 Representação da solução

Dado o domínio de utilização das metaheurísticas propostas, é natural inferir que uma solução seja formada estritamente por um AC. Do ponto de vista computacional, um AC pode ser entendido como uma estrutura n-dimensional discreta em relação ao tempo/espaço composta por um conjunto de elementos menos complexos. Uma forma bem simples de reproduzir essas estruturas consiste na utilização de vetores n-dimensionais. Um vetor é usado para representar as células do AC e o outro vetor usado para representar o conjunto de regras correlato, em um arranjo parecido com o retratado na Figura 4.2.

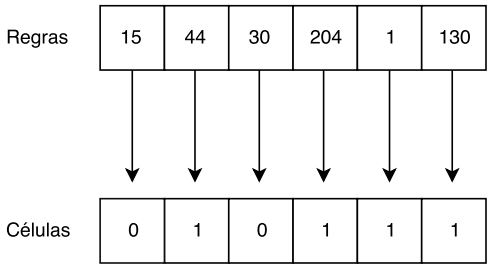

Figura 4.2: Representação vetorial de um AC

Baseando-se nos conceitos apresentados na seção 2.1, optou-se por utilizar ACs unidimensionais binários heterogêneos submetidos a uma vizinhança de raio 1 representados conforme descrito acima. Uma certa flexibilização no conceito de vizinhança apresentado na seção 2.2 também fez-se necessária, já que nessa representação pode não ser possível delimitar com precisão todos os elementos pertencentes a vizinhança de uma célula próxima a uma das extremidades do AC. Para contornar essa restrição foi utilizado um tratamento de borda do tipo circular nestes elementos. Com isso, por exemplo, a primeira célula passa a ter como vizinha a última célula e vice-versa. A Figura 4.3 fornece uma noção geral do resultado da aplicação desse procedimento.

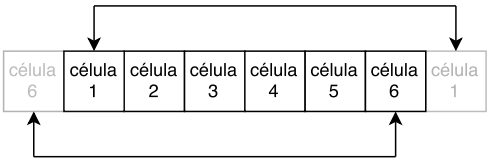

Figura 4.3: Condições de contorno circular em um AC.

Geralmente, tem-se o conjunto de regras representado como um vetor de números inteiros, porque existe uma correspondência entre o comportamento da regra com sua representação binária. Por exemplo, é possível explicitar toda configuração possível de vizinhança e o resultado da aplicação da regra 90, por meio da análise da representação binária desse número, conforme demonstrado nas Figura 4.4 e 4.5.

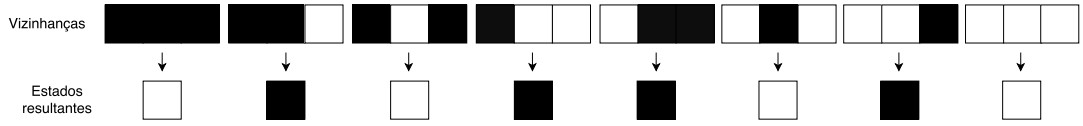

Figura 4.4: Regra 90

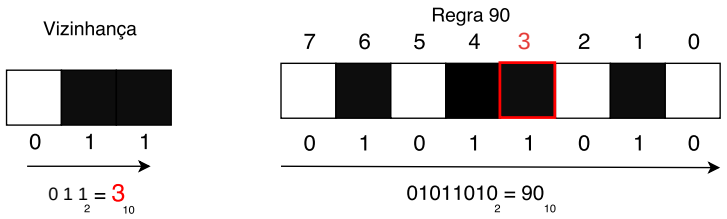

Figura 4.5: Vizinhança x Regra 90

4.3 Metaheurísticas

Partindo dos objetivos propostos na seção 1.3, foram definidas as metaheuristicas que seriam aplicadas na solução do problema exposto inicialmente na seção 1.1. As metaheurísticas selecionadas foram: o SA, a Busca Tabu e o ACO.

Com isso, quatro versões do algoritmo principal foram desenvolvidas. A primeira, usa somente a versão clássica do SA, a segunda versão combina o SA com um mecanismo probabilístico para melhorar a escolha das regras durante as iterações, a terceira versão

combina o SA com o mecanismo probabilístico e a Busca Tabu e por fim, uma versão criada usando a implementação clássica do ACO.

4.4 Cálculo de Entropia

A função objetivo utilizada pelas metaheurísticas foi a entropia, definida na equação (2.2), onde cada evento E_i representa uma subsequência diferente que pode ser gerada por um autômato celular M, e o valor n é o número máximo de subsequências diferentes que podem ser geradas a partir desse mesmo autômato. Este valor é calculado com base no tamanho $|M|$ e no número de estados que o automato usado pode assumir Q, num total de $Q^{|M|}$ variações possíveis.

O procedimento desenvolvido para cálculo de Entropia é bem simples. Inicialmente, o AC é evoluído por n passos e a frequência associada a cada configuração diferente gerada é armazenada. O cálculo descrito na equação 2.2, então, é realizado com base nessas frequências armazenadas. Algumas alterações tiveram de ser implementadas com o intuito de otimizar todo este procedimento. A modificação mais significativa tem por objetivo evitar que o AC seja evoluído de forma desnecessária quando a chave gerada pelo mesmo já contiver um ciclo. Uma vez identificado esse ciclo, basta calcular quantos ciclos cabem no que resta de gerações a serem geradas. Com isso, é possível fornecer uma previsão bem precisa da frequência dos elementos restantes sem que seja necessário evoluir o AC por esses passos restantes.

4.5 SA Clássico

Duas versões do SA Clássico foram utilizadas (Brito et al, 2017). A diferença entre as mesmas está no critério adotado para verificação do equilíbrio térmico em uma dada temperatura. Na primeira versão, denotada “SA1”, o equilíbrio térmico é dado por um número de iterações do algoritmo sem melhora no valor da função objetivo. Na segunda versão (“SA2”), o equilíbrio térmico ocorre quando a média aproximada da função objetivo chega a um estado onde as variações geradas são bem pequenas e próximas ao seu valor máximo e exato.

Para determinar quando o sistema "cristaliza", é criado um ponto de temperatura mínima do sistema (T_{min}). Segue uma descrição detalhada do algoritmo proposto:

1. É gerada uma solução inicial s_0 formada por um autômato celular e seu conjunto de regras correspondente, ambos gerados aleatoriamente.

2. Uma temperatura inicial elevada T_0 é associada a s_0. É definido o critério utilizado para verificar o equilíbrio térmico em uma dada temperatura.

   (a) Se o critério usado for o número de iterações sem melhora na função objetivo (SA1), é criado um contador $Iter$ para guardar esse número e outro valor $IterMax$ é definido como limite para $Iter$.

   (b) Se o critério usado for a estabilização da média da função objetivo (SA2), é definido um número mínimo de iterações por temperatura.

3. O autômato é evoluído por n passos a partir da aplicação do conjunto de regras de transição. Este valor corresponde ao número máximo de subsequências que podem ser geradas sem repetição por um autômato do tamanho utilizado. Por fim, calcula-se a entropia a partir desse número de passos.

4. Uma das regras do autômato é sorteada aleatoriamente e trocada por outra (entre 0 e 255), gerando uma solução corrente s. O autômato é novamente evoluído, só que com essa nova regra no conjunto e a entropia é recalculada.

5. É calculada a variação Δf entre o valor da entropia calculada antes da regra ser alterada e o valor da entropia calculada depois que a regra foi alterada.

   (a) Se $\Delta f \geq 0$, s é aceita como nova solução a partir desse instante. O conjunto de regras do autômato passa a conter a regra alterada.

   (b) Se $\Delta f < 0$ for negativo, s pode ser ou não ser aceita como nova solução. Para determinar isso, é calculada a probabilidade $p = e^{\frac{-d_z}{KT}}$. Um número aleatório (entre 0 e 1) é sorteado. Se o número cair dentro do intervalo entre 0 e p, a regra alterada permanece no conjunto de regras do autômato e s é aceita como nova solução. Caso contrário, o valor original da regra alterada é restabelecido e a solução s é descartada.

6. O valor $Iter$ passa por uma atualização se o critério usado para equilíbrio térmico for o número de iterações sem melhora na função objetivo.

7. Verifica-se se o sistema atingiu o equilíbrio térmico.

   (a) No primeiro caso (SA1) é verificado se o valor de $Iter$ é igual ao valor $IterMax$.

      i. Se o valor for igual, a temperatura atual tem um leve decréscimo dado por α e o algoritmo retorna ao passo 4.

      ii. Se o valor for diferente, retorna-se ao passo 4 sem mudar a temperatura.

   (b) No segundo caso (SA2) é necessário calcular a diferença das médias da entropia atual e da entropia há $IterMinTemp$ (número mínimo de iterações por temperatura) atrás.

      i. Se a diferença for menor que o valor ϵ, a temperatura sofre um pequeno decréscimo dado por α e o algoritmo retorna ao passo 4.

      ii. Do contrário, retorna-se ao passo 4 sem mudar a temperatura.

8. O fim do processo ocorre quando a temperatura chega a um valor muito próximo de 0, T_{min}, que indica que o sistema está cristalizado.

Os parâmetros usados descritos a seguir foram obtidos a partir de testes empíricos: Temperatura inicial (T_0) = 0,1; Decréscimo da temperatura (α) = 0,98; Temperatura mínima (T_{min}) = 0,001; Mínimo de iterações por temperatura ($IterMinTemp$) = 200; Diferença das médias (ϵ) = 0,01.

O número máximo de iterações ($IterMax$) é um parâmetro cujo valor inicial foi obtido a partir da equação $T_n = \alpha T_{n-1}$, . Chegando a um valor $m = \log_\alpha \frac{T_{min}}{T_0}$, onde α, T_{min} e T_0 são, respectivamente, o decréscimo da temperatura, a temperatura inicial e a temperatura mínima usadas como parâmetro no SA. O valor inicial desse parâmetro foi definido como 230 iterações, porém este valor foi alterado para 500 iterações depois da realização de alguns testes. Este número se mostrou bem razoável em temperaturas mais baixas quando o algoritmo aplica busca local nas soluções e em temperaturas mais altas quando o algoritmo está em sua fase de explotação.

4.6 SA Clássico e mecanismo probabilístico (Roleta)

Na abordagem clássica do SA, todas as regras têm uma mesma probabilidade de escolha. Porém, como já foi observado em alguns trabalhos da literatura (Tomassini e Perrenou,

2001; Villela e Carvalho, 2007), e depois de um número suficiente de testes, verificou-se que só um pequeno número de regras leva a soluções boas. Assim, um mecanismo probabilístico é usado para dar maior probabilidade de escolha às regras que levaram a resultados melhores em iterações ou execuções anteriores.

Para tanto, foi criado um sistema de seleção de regras bem similar a seleção por roleta presente em muitos algoritmos evolutivos (Goldbarg et al, 2015). Esse mecanismo é modelado por meio de uma matriz bidimensional. As linhas da matriz são as regras presente no conjunto de regras e as colunas representam cada célula do autômato utilizado. Com isso, uma regra terá uma probabilidade diferente em cada célula do autômato. Visto que esse é um problema onde não só a regra é importante, mas também a posição onde essa regra aparece. A matriz armazena o número de vezes em que uma regra apareceu em uma determinada posição para um alvo definido. Este alvo tem um valor um pouco inferior ao valor máximo da função objetivo obtida até então na execução atual ou de uma execução anterior, quando a matriz é importada de outra execução. Todas as regras das soluções que estiverem no intervalo entre o valor alvo e o valor máximo da função objetivo são armazenadas na matriz na posição em que aparecem na solução. No cálculo da regra que será trocada em uma dada posição, a matriz é convertida em uma matriz de probabilidade e a função aleatória passa a dar mais peso à regra com maior probabilidade na posição informada, mecanismo denotado neste trabalho como Roleta. Esse mecanismo está ilustrado nas figuras 4.6 e 4.7.

A roleta permite que o algoritmo seja direcionado às melhores regiões do espaço de busca. Um novo alvo é definido toda vez que o valor da função objetivo é melhorado, esta atualização é feita da seguinte maneira: $alvo = \frac{melhorEntropia}{maxEntropia} * \sigma$, onde $melhorEntropia$ é o valor máximo de entropia obtido até então, $maxEntropia$ é o máximo global da função de entropia e σ é o fator de proporcionalidade, fixado em 0,95. Um valor inicial é fixado para o alvo em situações onde não a matriz não é pré-carregada com informações de execuções anteriores. O valor inicial definido para o alvo corresponde a 45% do valor máximo da função de entropia ($maxEntropia$).

Testes preliminares mostraram que esta abordagem foi capaz de atuar evitando a convergência precoce do algoritmo em algumas execuções. Porém, em certos casos,

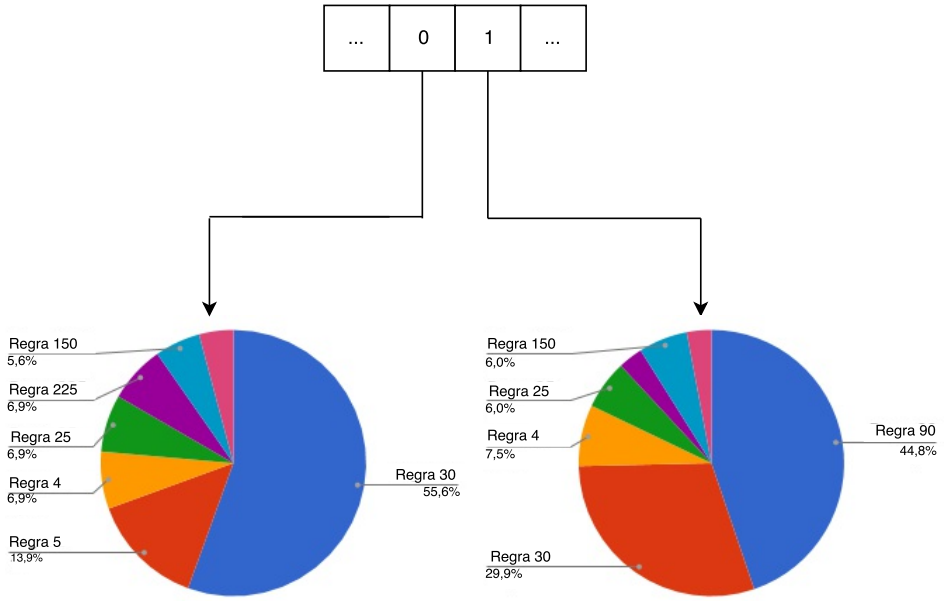

Figura 4.6: Representação das roletas para duas células do AC

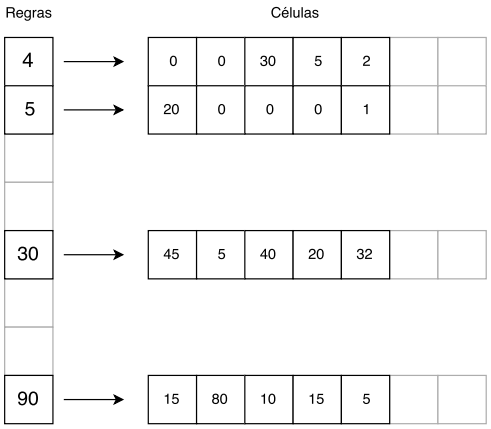

Figura 4.7: Matriz Regras x Células

principalmente em regiões com um número grande de soluções homogêneas ou próximo de muitos ótimos locais, houve uma inversão deste comportamento e a roleta passou a viciar, fazendo com que o algoritmo fosse levado a uma convergência precoce para valores distantes do ótimo. Isso foi observado em diversos casos e pode ser visto com mais detalhes na seção de resultados.

4.7 SA Clássico e Roleta combinados com a Busca Tabu

Para minimizar o impacto da convergência precoce do algoritmo com o uso da Roleta, optou-se por inserir um módulo de Busca Tabu na abordagem. Com sua política de exploração extensiva da vizinhança de uma solução, objetivou-se sanar o problema. A ideia central é que a Busca Tabu atue periodicamente sobre soluções geradas pelo SA, como forma de evitar um aumento considerável na complexidade do algoritmo. A Busca Tabu é executada por um número fixo de 1000 iterações sem que haja uma melhora na função objetivo.

Na abordagem proposta, foram empregadas as ideias de memória adaptativa (lista tabu) e estratégia de busca baseada em memória do algoritmo padrão da Busca Tabu. A lista tabu usada suporta 100 movimentos. O tamanho da lista foi definido depois de testes empíricos e os movimentos tabu são aqui representados por trocas de regras em determinadas posições. Toda vez que a Busca Tabu é chamada, ao invés de sortear uma regra e uma posição baseada na Roleta, somente a regra é sorteada.

A partir de uma solução obtida, a vizinhança desta nova solução é analisada por meio da aplicação da regra sorteada em cada célula do autômato. Em cada troca, a entropia é calculada e armazenada em um vetor e a regra anterior é restabelecida à sua posição. Depois de todas as trocas, o vetor é ordenado de forma decrescente e a primeira posição que terá ao movimento que levou ao maior valor de entropia, dentre os calculados, é escolhida como solução corrente. O movimento para chegar a essa solução é armazenado na lista tabu e se torna um movimento proibido.

O critério tabu dos movimentos deve ser respeitado. Por isso, se o movimento escolhido como o melhor for um movimento proibido (tabu), ele poderá ser descartado como solução e o próximo movimento, que representa o segundo maior valor obtido, será escolhido, e assim sucessivamente. O movimento tabu não é sumariamente descartado, pois, se ele melhorar a função objetivo ou se todos os movimentos disponíveis forem Tabu, ele é escolhido mesmo assim (função de aspiração). O processo pode ser entendido com mais clareza a partir do seguinte algoritmo:

1. Uma regra é sorteada aleatoriamente (entre 0 e 255).
2. A regra sorteada é associada a uma célula do autômato.
3. A entropia é calculada usando o novo conjunto de regras e armazenada em um vetor.
4. A regra que foi trocada pela regra sorteada é devolvida à célula correspondente.
5. É feita a verificação se toda vizinhança da solução foi analisada.
   (a) Se a vizinhança não foi toda analisada, o passo 2 do algoritmo é repetido para uma nova célula do autômato. Isso é feito porque há trocas a serem realizadas.
   (b) Se toda a vizinhança foi analisada, o próximo passo pode ser realizado.
6. O vetor com as entropias é ordenado de forma decrescente. A solução correspondente ao movimento encontrado na primeira posição pode ser escolhida como solução corrente.
   (a) O movimento é escolhido como solução corrente se não for um movimento Tabu, ou se os critérios da função de aspiração não forem atendidos (melhora na função objetivo ou se todos os movimentos forem Tabu).
   (b) Se nenhuma dessas situações ocorrer, o passo 6 é repetido só que com a possível solução sendo a posição seguinte do vetor.
7. O movimento escolhido é adicionado à lista Tabu. Se a lista estiver cheia, o movimento mais antigo é retirado.
8. Se o critério de parada for atingido, a solução final é passada para o SA.

4.8 ACO

O ACO, introduzido na seção 2.4.3, apareceu como uma alternativa às metaheurísticas de solução única descritas nas seções anteriores deste capítulo. Almeja-se, com o uso do ACO, uma possível análise do comportamento de metaheurísticas dessa natureza frente a problemática apresentada e a outros métodos já especificados.

A versão do ACO implementada é baseada na versão mais conhecida dessa metaheurística introduzida em (Dorigo e Di Caro, 1999). Para adaptar a metaheurística ao problema, fez-se necessária a construção de um grafo conforme descrito na seção 2.4.3.

Esse grafo deve modelar de forma satisfatória o espaço de soluções explorado de um problema durante a execução da metaheurística. Neste sentido, optou-se pelo desenvolvimento de um grafo $G(V, E)$ bipartido completo. As partições desse grafo são compostas, respectivamente, pelas regras e pelas posições possíveis dessas regras dentro do AC como demonstrado na Figura 4.8. As arestas desse grafo são caminhos virtuais para as formigas que depositam seus feromônios nesses caminhos no decorrer do algoritmo.

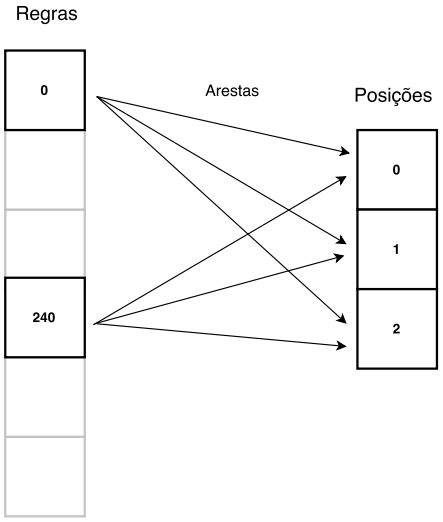

Figura 4.8: Grafo bipartido associando regras e posições

O processo de atualização dos feromônios nas arestas é dado por meio das equações 4.2 e 4.3. Esse processo de atualização é complementado por um reforço adicional nas arestas pertencentes a melhor solução, segundo a equação 4.4. Um outro passo bem importante nessa abordagem é o de seleção de arestas que pertencerão a solução recém-construída por uma formiga. Esse processo de seleção é fundamentado num mecanismo de roleta bem semelhante ao mecanismo que foi incorporado ao SA na seção 4.6. A diferença entre os mecanismos está na função densidade de probabilidade relacionada, que nesse caso é dada pela equação 4.5, e no fato de que nessa abordagem, uma única roleta é associada a todas as arestas do grafo. As equações utilizadas pelo ACO são:

$$\text{Evaporação: } \tau_{ij} = (1 - \rho)\tau_{ij} \quad 0 \leq \rho \leq 1 \tag{4.2}$$

$$\text{Reforço: } \tau_{ij} = \tau_{ij} + \Delta_{ij}^{k} \tag{4.3}$$

$$\text{Reforço Melhor solução: } \tau_{ij} = \tau_{ij} + \Delta_{ij}^{bs} \tag{4.4}$$

$$p_{ij} = \begin{cases} 0 & \text{aresta } ij \text{ visitada} \\ \frac{(\tau_{ij})^{\alpha}.(\eta_{ij})^{\beta}}{\sum_{k=0}^{n} p_{ik}} & \text{aresta } ij \text{ não visitada,} \end{cases} \tag{4.5}$$

onde τ_{ij} são os feromônios associados à aresta ij, η_{ij} é o valor heurístico extraído, que nesse caso é o valor da função de Entropia obtido ao adicionar essa aresta a solução, ou seja, o valor encontrado ao atribuir a regra em questão a posição definido pelo vértice. Os valores α e β são elementos responsáveis, respectivamente, por dar um maior peso no cálculo ao feromônio ou a informação heurística. O valor Δ_{ij}^{k} representa o custo da solução da formiga k se a aresta ij pertencer a essa solução e o valor Δ_{ij}^{bs} representa o custo da solução da formiga com a melhor solução se a aresta ij pertencer a essa solução. Os custos envolvidos são os valores de entropia extraídos por meio das soluções presentes nas formigas.

Um fator ξ pode ser adicionado a equação 4.3 para controlar o processo de reforço dos feromônios. O mesmo se aplica a equação 4.4, por meio da inclusão de um fator extra ξ_{bs}.

Os valores usados como parâmetros do algoritmo, descritos a seguir, foram obtidos em (Dorigo e Stützle, 2009) e por meios de testes empíricos: Taxa de decréscimo no feromônio (ρ) = 0.4; Peso no feromônio (α) = 0.2; Peso na heurística (β) = 0.8; Fator de reforço no feromônio (ξ) = 2; e Fator de reforço no feromônio na melhor solução (ξ_{bs}) = 5. O número de formigas é dado pelo tamanho do autômato usado.

Assim como no algoritmo original, definiu-se como critério de parada um número máximo de iterações ($IterMax = 10000$). O ACO implementado está descrito a seguir:

1. Uma geração inicial de formigas é criada de forma randômica com base em um AC M. Cada formiga terá um AC associado com a mesma configuração inicial de M, porém com um conjunto de regras possivelmente diferente.

2. Os feromônios nas arestas do grafo são inicializados com um valor bem pequeno (0.001) e logo em seguida é promovida uma atualização (evaporação e reforço) nesses

mesmos feromônios fundamentada na primeira geração de formigas.

3. Uma nova geração de formigas é gerada a partir do seguinte procedimento:

   (a) Cada formiga da nova geração constrói sua solução por meio da solução gerada pela formiga da geração anterior. Neste processo é selecionada uma aresta pertencente ao grafo usando o mecanismo de seleção descrito anteriormente. Essa aresta é adicionada a solução substituindo uma outra aresta pertencente a solução na mesma posição. A Entropia dessa nova solução é calculada.

   (b) O passo anterior repete-se até que ocorra uma melhora na função objetivo associada a formiga ou por um número máximo de iterações definido ($IterConstrMax = 3$). Nessas condições, entende-se que a construção da solução da formiga terminou e o processo pode ser repassado a outra formiga.

4. Os feromônios são atualizados com base nessa nova geração de formigas.

5. Se o critério de parada não for atendido, repete-se o passo 3.

5 Experimentos e Resultados

Este capítulo apresenta os resultados obtidos executando cada um dos algoritmos mencionados no Capítulo 4, a fim de validar os métodos no problema de geração de autômatos celulares caóticos aplicáveis como mecanismo geradores de chaves de fluxo. A ideia é que alguns possíveis cenários de utilização dessas abordagens sejam simulados, com o intuito de tentar observar características do problema previstas durante o desenvolvimento deste trabalho, e que de alguma forma auxiliem o processo de convergência dos algoritmos implementados. Os experimentos que envolvem o fator tempo foram realizados em um Athlon II X2 245 2.90GHZ com 4GB DDR2 800MHZ em um sistema operacional Ubuntu 14.10. Os algoritmos foram implementados por meio da linguagem C++.

No âmbito criptográfico existem determinadas aplicações em que é mais interessante obter uma solução sub-ótima em um período de tempo mais curto (Villela e Carvalho, 2007). Neste contexto, foi proposta uma comparação entre os algoritmos fundamentada em uma análise probabilística, que relaciona o tempo de processamento à probabilidade acumulada de convergência dos algoritmos. Assim, pode-se observar, em um determinado tempo de processamento, a probabilidade da convergência do algoritmo para um valor alvo preestabelecido para função objetivo.

Nesse experimento foram comparadas as duas abordagens clássicas e as duas abordagens combinadas à Roleta. Os testes foram definidos sobre três alvos, que correspondem a 50%, 75% e 90% do valor máximo da entropia. Como já foi dito, a simples combinação entre Roleta e os SAs apresenta problemas em situações próximas a muitos ótimos locais e à regiões com muitas soluções homogêneas próximas à melhor solução encontrada até o momento. Uma solução inicial foi gerada a partir de um autômato com configuração inicial {1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0} e um conjunto de regras {91, 208, 189, 242, 223, 168, 200, 68, 162, 34, 115, 229, 155, 222, 114, 183} cujo valor de entropia foi de 1,00125. Partindo desta solução, o algoritmo foi executado 50 vezes para os alvos de 50% e 75% e 20 vezes para o alvo de 90%. O algoritmo é parado quando o alvo é atingido e o tempo necessário é contabilizado.

A Figura 5.1-(a) mostra que, para obter soluções com entropia a 50% do ótimo, as duas abordagens que usam Roleta sempre têm chance de sucesso antes das abordagens puras. Ao tomar um alvo mais difícil, como o apresentado na Figura 5.1-(b), o comportamento se mantém, mas observa-se uma maior eficácia das abordagens com Roleta, que têm uma probabilidade maior de alcançar o alvo definido num intervalo de tempo menor.

Para a entropia em 90% como alvo, foi feita uma alteração no algoritmo, de forma que a Roleta só seja utilizada até que o algoritmo obtenha uma solução com entropia de 60% do valor máximo. Este valor foi definido observando testes empíricos e o gráfico da Figura 5.1-(b), onde é visto que as abordagens que combinam Roletas já começam a apresentar um comportamento deficiente a partir de um certo tempo, quando as curvas referentes às abordagens com Roleta ultrapassam as curvas referentes às abordagens puras, indicando que estas apresentam maior probabilidade de alcançar o alvo.

Os gráficos das Figuras 5.1-(a), 5.1-(b) e 5.1-(c) revelam uma maior tendência das abordagens com roleta de encontrar as soluções alvo em um intervalo de tempo menor que as abordagens simples, o que torna essas abordagens mais interessantes para aplicações que necessitem de soluções num intervalo de tempo mais curto. Além disso, tornou-se evidente um fato que estava sendo defendido: a Roleta consiste em uma abordagem eficiente no que se refere ao direcionamento do algoritmo a regiões promissoras dentro do espaço de soluções do problema, uma espécie de partida inteligente para o algoritmo. A análise dos gráficos serviu ainda como base para definir quais versões desenvolvidas possivelmente teriam melhor comportamento se combinadas à Busca Tabu. As abordagens selecionadas foram as combinações com a Roleta, dando origem a duas novas versões desses algoritmos com um módulo de Busca Tabu anexada, conforme descrito na seção 4.7.

Com as versões finais dos algoritmos definidas, foi realizado um experimento com o propósito de comparar essas abordagens por meio de 3 cenários definidos. Nesse experimento todos os algoritmos foram executados 30 vezes e em cada uma dessas execuções foi utilizada uma semente de randomização diferente. No primeiro cenário, foram utilizados ACs de 8 bits, assim como os usados por Villela e Carvalho (2007). Num segundo cenário, os algoritmos foram submetidos a ACs de 12 bits e por fim, num último cenário, submetidos a ACs maiores compostos por 16 bits, mais próximos de aplicações reais. Ao

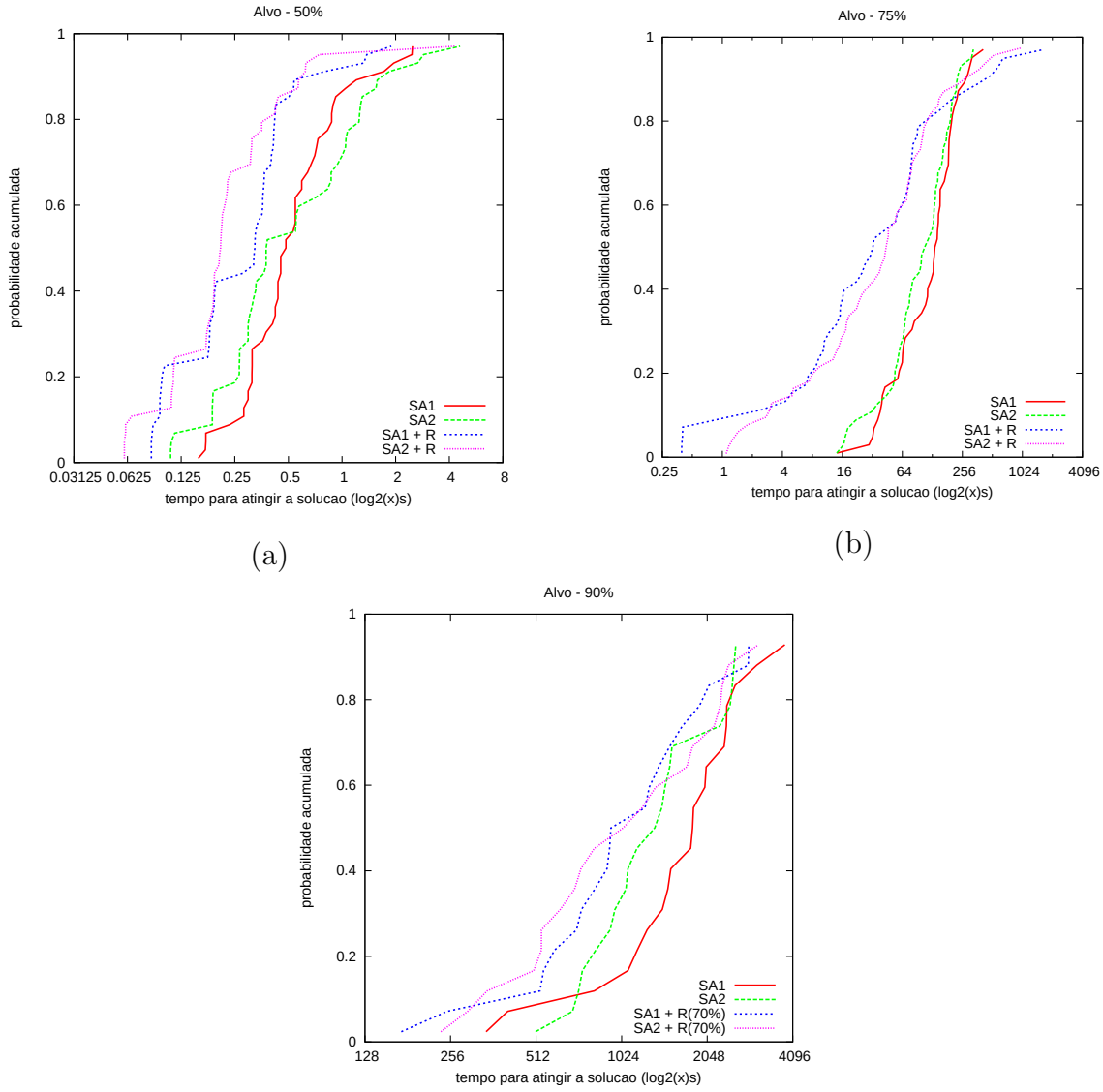

Figura 5.1: Gráficos probabilísticos (Convergência x Tempo)

final de cada execução anotou-se a melhor solução obtida e seu valor de Entropia. O ACO foi executado somente no primeiro cenário, porque foi observado que o mesmo não se mostrou competitivo em comparação às outras abordagens desenvolvidas.

Os parâmetros utilizados em todos os cenários são os parâmetros descritos no capítulo 4. O único parâmetro variável entre os cenários é o $IterChamadaTabu$, número de iterações máximo sem melhora na função objetivo necessário para acionar a Busca Tabu, que foi definido para 10000 iterações no primeiro cenário e 20000 iterações nos outros dois cenários. Os resultados desse experimento estão descritos nas Tabelas 5.1, 5.2 e 5.3 e nos gráficos das Figuras 5.2 a), 5.2 b) e 5.2 c).

No primeiro cenário, fica evidente que os algoritmos, com exceção do ACO, conse-

Tabela 5.1: Resultados Cenário 8 bits

Abordagem Utilizada	**Valor Máximo**	**Valor Médio ± Desvio Padrão**	**Número de soluções (30 execuções)**		
			>80%	**>85%**	**>90%**
SA1	7,9922	7,3704 ± 0,8954	22	21	20
SA1+R (SA1R)	7,9922	7,3758 ± 0,8969	22	22	20
SA1+R+Tabu (SA1RTABU)	7,9922	7,7842 ± 0,3730	30	29	25
ACO	6,1556	5,5624 ± 0,2014	0	0	0
SA2	7,9922	7,4639 ± 0,8743	23	23	22
SA2+R (SA2R)	7,9922	7,4373 ± 0,8755	23	23	21
SA2+R+Tabu (SA2RTABU)	7,9922	7,8048 ± 0,3986	30	28	24

Tabela 5.2: Resultados Cenário 12 bits

Abordagem Utilizada	**Valor Máximo**	**Valor Médio ± Desvio Padrão**	**Número de soluções (30 execuções)**		
			>80%	**>85%**	**>90%**
SA1	11,9990	11,3937 ± 0.8937	29	29	28
SA1+R	11,9990	11,2798 ± 1,4881	27	27	26
SA1+R+Tabu	11,9995	11,6620 ± 0,4034	30	30	28
SA2	11,9995	11,3526 ± 0,9359	29	28	25
SA2+R	11,9995	10,8001 ± 1,8712	24	24	22
SA2+R+Tabu	11,9995	11,7878 ± 0,3648	30	30	29

guiram atingir o ótimo global da função de Entropia, porém percebe-se uma superioridade nas abordagens com Tabu, que conseguiram obter um número maior de soluções dentro dos limites de 80%, 85% e 90% do valor máximo de Entropia e apresentaram uma dispersão menor nos valores de Entropia obtidos a partir de execuções independentes dos algoritmos. A convergência precoce do algoritmo associada a problemas retratados com as abordagens baseados no uso somente da Roleta ficou caracterizada por uma dispersão maior nos dados gerados a partir dessas abordagens, conforme demonstrado no gráfico da Figura 5.2 a). As abordagens mais básicas, nesse cenário, também sofreram com problemas de convergência precoce, conforme pode ser visto, por exemplo, no número

Tabela 5.3: Resultados Cenário 16 bits

Abordagem Utilizada	**Valor Máximo**	**Valor Médio ± Desvio Padrão**	**Número de soluções (30 execuções)**		
			>80%	**>85%**	**>90%**
SA1	15,9999	14,8762 ± 0,9646	29	26	21
SA1+R	15,9828	14,5531 ± 1,6422	28	27	19
SA1+R+Tabu	15,9999	15,4759 ± 0,5488	30	30	29
SA2	15,9808	14,8579 ± 0,8646	30	27	24
SA2+R	15,9999	14,9384 ± 1,4355	27	26	25
SA2+R+Tabu	15,9919	15,4219 ± 0,5234	30	30	29

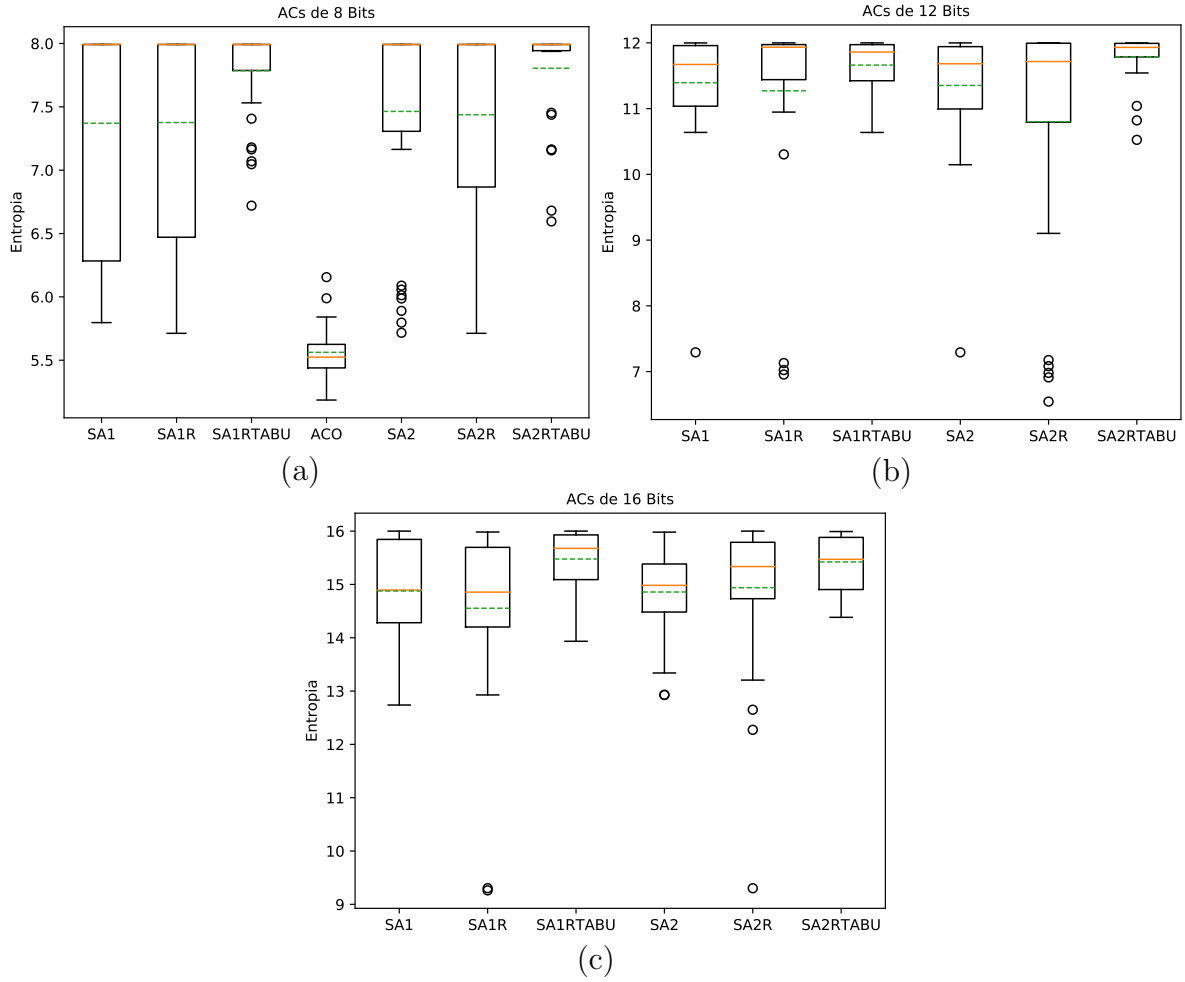

Figura 5.2: Gráficos *Boxplot* Entropia - (Média em verde)

considerável de *outliers* abaixo de um limite de 75% do valor máximo de Entropia no *Boxplot* referente ao SA2 e numa maior dispersão dos dados presente no *Boxplot* referente ao SA1.

Com base nos resultados deste primeiro cenário, optou-se por retirar o ACO dos restantes dos experimentos, visto que os resultados obtidos a partir dele ficaram aquém do esperado, indicando que essa abordagem precisa passar por alterações para se tornar competitiva se comparada às outras abordagens desenvolvidas.

No segundo cenário, percebe-se novamente uma superioridade nas abordagens com Tabu, porém vale salientar que as abordagens simples também apresentaram resultados bem equiparáveis as abordagens com Tabu, levando em consideração o número de soluções geradas dentro dos limites definidos. Nesse cenário, percebe-se que as abordagens com Roleta também sofreram com problemas referente a convergência precoce, conforme pode ser visto de forma mais acentuada no *Boxplot* referente ao SA2R e de forma mais

branda no *Boxplot* do SA1R. Um fator bem importante a ser ressaltado é que as versões SA1 e SA1R não conseguiram atingir o ótimo global da função de Entropia, apesar de apresentarem um valor médio superior ao encontrado em seus correspondentes SA2 e SA2R.

No último cenário, ficam mais perceptíveis as contribuições da Busca Tabu no processo de convergência dos algoritmos que se beneficiaram dela. O número de soluções geradas dentro dos limites definidos, o valor médio associado ao desvio padrão e uma análise dos *BoxPlots* referentes às duas abordagens são ótimas indicações disso e resumem o que foi defendido neste trabalho: a Busca Tabu seria capaz de minimizar o impacto da convergência do algoritmo com o uso da Roleta. Uma observação importante a ser mencionada é que somente as versões SA1, SA1RTABU e SA2R atingiram o ótimo global da função objetivo. Nesse cenário, os efeitos da convergência muito precoce se manifestaram de forma mais branda nos algoritmos.

Uma outra análise comparativa foi proposta, mais direcionada para comparação entre as abordagens desenvolvidas com trabalhos relacionados presentes na literatura. Tomassini e Perrenou (2001) e Wolfram (2002) apresentam, em um desses trabalhos, um conjunto definido de regras com comportamento caótico que, em tese, seriam soluções em potencial para a problema apresentado na seção 1.1. Tomassini e Perrenou (2001) delimitaram um conjunto formado pelas regras {90, 105, 150, 165} que seriam capazes de assegurar o comportamento caótico de um AC. Wolfram (2002) apresentou a regra 30 como caótica e explicitou as suas principais características como mecanismo gerador de números aleatórios. Dessa forma, foi proposto um experimento com o intuito de comparar os valores de Entropia obtidos por meio das abordagens propostas, apresentadas no experimento anterior, e os valores obtidos a partir de 30 execuções de ACs gerados com base na regra 30 e de permutações geradas com base nas regras de Tomassini. Nesses testes foram usados os mesmos parâmetros definidos para o experimento anterior. Os valores obtidos nas execuções estão na Tabela A.3.

Do ponto de vista do valor da função de Entropia e com base na análise do gráfico 5.3, fica evidente a superioridade das abordagens propostas dentro do escopo proposto em relação às soluções geradas com base no conjunto de regras de Tomassini e da regra

30. Vale ressaltar que a regra 30 têm um valor de Entropia razoável, porém a utilização dela pode diminuir a complexidade do sistema como um todo, uma vez que se têm conhecimento do algoritmo utilizado e que sabe-se que o conjunto de regras é homogêneo e é composto somente pela regra 30. Com isso, o esforço de um sistema invasor será direcionado exclusivamente para encontrar qual configuração inicial foi utilizada no AC's que criptografaram um determinado trecho da mensagem. As soluções geradas com base nas regras de Tomassini ficaram com o pior desempenho frente às outras abordagens, comportamento já observado e apresentado nos resultados do trabalho de Villela e Carvalho (2007).

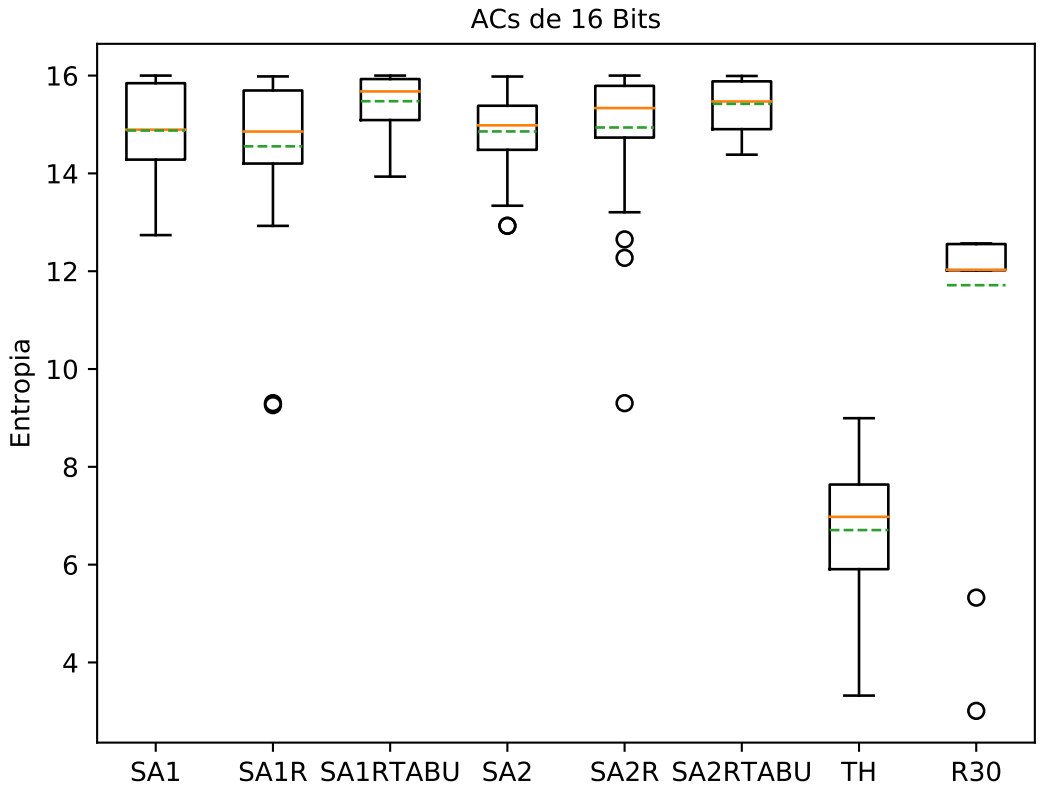

Figura 5.3: Abordagens x Regras Tomassini (TH) e Regra 30 (R30) - *Boxplot*

A validação das abordagens propostas como soluções para o problema apresentado depende ainda de uma avaliação mais substancial da qualidade das soluções geradas pelo mesmo. Uma forma de realizar essa avaliação consiste em tratar uma solução gerada como um mecanismo gerador e submetê-la a testes bem conhecidos na literatura. Esses

testes são capazes de mensurar a eficácia desses mecanismos no domínio criptográfico. No presente trabalho, optou-se pela utilização da bateria de testes do NIST (*National Institute of Standars and Technology*), cujo principal objetivo é atestar o potencial criptográfico desses geradores, por meio de 15 testes que verificam os tipos de não-aleatoriedades mais comuns nesses elementos. Uma descrição completa dos testes pode ser encontrada na Tabela B.1. Uma sequência de bits geradas por meio de uma solução de valor de entropia máximo, obtida a partir dos experimentos anteriores, foi submetida a essa bateria de testes. Outras duas sequências de bits foram geradas, respectivamente, a partir da melhor solução obtida por meio da regra 30 e da melhor solução obtida por meio da combinação de regras de Tomassini e Perrenou (2001) e também foram submetidas a este mesmo teste para fins comparativos. Os ACs correspondentes a essas três soluções e o seu valor de entropia estão apresentados na Tabela 5.4.

Tabela 5.4: Configurações inicias com os valores correspondentes de entropia

Gerador	**Configuração Inicial**	**Conjunto de Regras**	**Entropia**
Abordagens Propostas	0101001000101110	{51, 60, 108, 102, 165, 165, 165, 165, 165, 90, 90, 105, 150, 150, 90, 165}	15,9999
Regras Tomassini	1110011010010111	{150, 150, 165, 150, 165, 165, 165, 150, 105, 90, 150, 150, 165, 105, 90, 165}	8,76155
Regra 30	0001101011101011	{30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30}	12,5667

Foram executados 13 dos 15 testes. Os testes de Posto de Matriz Binária e Transformada Discreta de Fourier não foram executados, pois o número de bits da sequência não é adequado para esses testes. Nos testes o valor obtido como retorno (p-valor) varia entre 0 e 1 e deve ser maior que 0,01 para que a sequência seja considerada aleatória. Segundo Bassham et al (2010), um p-valor $\geq 0,01$ indica que a sequência usada é considerada randômica com uma confiança de 99,9%. O tamanho da sequência gerada é de 1048256 bits. Os resultados dos testes são apresentados na Tabela 5.5 e ratificam a viabilidade do método dentro do domínio proposto. Vale ressaltar que os resultados encontrados nas Tabelas 5.6 e 5.7 corroboram a suposição de que ACs ditos caóticos gerados sem um processo de verificação, podem não ser aplicáveis ao ambiente criptográfico. Os parâmetros usados nos testes estão descritos na Tabela B.2.

Tabela 5.5: Resultados dos testes do NIST - Abordagem

Teste	***p*-valor**	**Resultado**
Entropia Aproximada	1,000000	Passou
Frequência em Blocos	0,495684	Passou
Somas Cumulativas	0,719266 e 0,713760	Passou
Frequência	0,995325	Passou
Complexidade Linear	0,523471	Passou
Corrida Mais Longa	0,588317	Passou
Não Sobreposição de Padrão	0,012360 a 0,992700	Passou
Sobreposição de Padrão	0,416234	Passou
Excursões Aleatórias	0,146189 a 0,804435	Passou
Variante de Excursões Aleatórias	0.016873 a 0,936061	Passou
Corridas	0,992208	Passou
Serial	1,000000	Passou
Universal	0,355272	Passou

Tabela 5.6: Resultados dos testes do NIST - Tomassini e Perrenou (2001)

Teste	***p*-valor**	**Resultado**
Entropia Aproximada	0,000000	Não Passou
Frequência em Blocos	0,000000	Não Passou
Somas Cumulativas	0,450536 e 0,454999	Passou
Frequência	0,240464	Passou
Complexidade Linear	0,000000	Não Passou
Corrida Mais Longa	0,000000	Não Passou
Não Sobreposição de Padrão	0,000000 a 0,042324	Não Passou
Sobreposição de Padrão	0,161045	Passou
Excursões Aleatórias	0,049806 a 0,703645	Passou
Variante de Excursões Aleatórias	0,073140 a 0,985983	Passou
Corridas	0,000000	Não Passou
Serial	0,000000	Não Passou
Universal	0,000000	Não Passou

Tabela 5.7: Resultados dos testes do NIST - Regra 30

Teste	***p*-valor**	**Resultado**
Entropia Aproximada	0,000000	Não Passou
Frequência em Blocos	1,000000	Passou
Somas Cumulativas	0,000000	Não Passou
Frequência	0,000000	Não Passou
Complexidade Linear	0,696181	Passou
Corrida Mais Longa	0,000000	Não Passou
Não Sobreposição de Padrão	0,00000 a 0.961729	Não Passou
Sobreposição de Padrão	0,161045	Passou
Excursões Aleatórias	0,000000	Não Passou
Variante de Excursões Aleatórias	0,000000	Não Passou
Corridas	0,000000	Não Passou
Serial	0,000000	Não Passou
Universal	0,000023	Não Passou

6 Considerações Finais

Esse trabalho propôs uma série de abordagens baseadas no uso de metaheurísticas para geração de autômatos celulares caóticos utilizados como mecanismo geradores de chaves de fluxo. Algumas características inerentes ao problema já observadas e descritas em outros trabalhos da literatura ficaram evidentes com o decorrer do desenvolvimento desse trabalho como: A alta característica combinatória associada a esse problema e regras que aparecem recorrentemente nas melhores soluções.

As alterações propostas nos métodos se mostraram eficientes dentro do propósito que foram sugeridas. A Roleta, por exemplo, se mostrou um ótimo mecanismo de partida inteligente para o algoritmo, já a Busca Tabu se mostrou uma alternativa bem eficiente como solução aos problemas de convergência presentes no algoritmo simples e nas abordagens com a Roleta. Vale ressaltar, no entanto, que a abordagem criada com base em uma metaheurística populacional, o ACO, não apresentou resultados satisfatórios. Acredita-se que esse fato têm relação com os parâmetros utilizados, que talvez não tenham sido os mais adequados ao tipo de problema que queria se solucionar. Além disso, existe ainda a possibilidade de que a estratégia adotada na adaptação do problema à abordagem não tenha sido a mais adequada, levando aos problemas de convergência observados nos experimentos.

Os experimentos serviram para validar as abordagens dentro do domínio estudado, possibilitando uma análise do comportamento dessas abordagens frente ao problema apresentado e além disso estimular um possível uso dessas abordagens em sistemas rodando em ambientes reais. Tendo como base o esquema descrito na Figura 2.2, por exemplo, pode-se sugerir um possível sistema criptográfico com base em uma das abordagens propostas. Esse sistema poderia ser constituído por um mecanismo gerador criado a partir de uma das abordagens com a Busca Tabu e uma função combinatória simples como o XOR apresentando na seção 2.1.1. Supondo que os ACs utilizados tenham um tamanho de 16 bits, que as chaves não serão reaproveitadas e que o tempo médio para gerar uma chave seja de 3379,78 segundos, conforme resultados obtidos por meio de 5 execuções

independentes de uma dessas abordagens e descritos na Tabela A.4, é possível criptografar um fluxo de bits de 145 bits/segundo. Esse valor é obtido estimando o tamanho da chave obtida com base no AC usado que é 1048576 bits e no tempo médio necessário para gerar uma nova chave previsto de forma pessimista em 2 horas (7200 segundos), fazendo com que não seja necessário utilizar mais que um ciclo da chave usada atualmente. Nesse modelo serão abstraídos detalhes referentes ao protocolo de comunicação utilizado, evitando questões como *transmission overhead* (Boosten, 1998) que reduziriam o fluxo de bits criptografado. Esse sistema poderia ser utilizado em uma aplicação bem simples como, por exemplo, um serviço de mensagens instantâneas. Supondo que a comunicação nesse sistema seja feita somente por mensagens de texto, que o usuário desse sistema não possa copiar/colar texto de outras fontes diretamente e com base no fluxo calculado de 145 bits/segundos, estima-se que nesse serviço de mensagem instantânea seja possível transmitir uma média de 18 caracteres(ASCII)/segundo. Uma taxa bem superior a taxa de digitação média de um usuário comum (Janela, 2014). Apesar de ser uma aplicação bem restrita, mostra-se que uma eventual utilização dessas abordagens em um ambiente real é possível. Aplicações mais complexas como VOIP necessitam de ACs maiores com 32, 64 bits para tornarem-se viáveis por meio das abordagens desenvolvidas, algo que é atualmente inviável dada a quantidade considerável de tempo necessária para obter uma solução desse tamanho.

Como proposta de trabalhos futuros, sugere-se a utilização de outras metaheurísticas para o problema, como, por exemplo, o ILS (*Iterated Local Search*) e o VNS (*Variable Neighborhood Search*), que podem ser combinados às metaheurísticas já desenvolvidas ou usadas como novas abordagens. Um outro a ponto a ser explorado pode ser a utilização de implementações mais robustas das metaheurística já utilizadas.

Além disso, sugere-se um estudo mais aprofundado de uma forma de tornar viável a aplicação do o ACO ou de outra metaheurística populacional, com o objetivo de estudar as implicações do uso de uma metaheurística dessa natureza dentro do domínio estudado.

Uma outra vertente a ser explorada é a possibilidade de calibrar melhor os parâmetros a serem usados nas versões desenvolvidas do algoritmo. Esta calibração, que atualmente é feita por meios de testes empíricos, pode ser realizada por meio de ferramen-

tas desenvolvidas especificamente para esta finalidade, como, por exemplo, o pacote irace (Bassham et al, 2010), que automatiza o processo de seleção de parâmetros buscando a melhor configuração a partir de um conjunto de instâncias informado para o problema.

A aplicação de paralelismo também pode ser um caminho a ser explorado como possível alternativa para diminuir o tempo e auxiliar no processo de convergência das abordagens.

Por fim, é levantada a possibilidade de criar uma "máquina" que seja capaz de criptografar e descriptografar mensagens com base nas chaves geradas pelo algoritmo desenvolvido, para observar o seu funcionamento em situações próximas das reais.

Bibliografia

Bassham, III, L. E.; Rukhin, A. L.; Soto, J.; Nechvatal, J. R.; Smid, M. E.; Barker, E. B.; Leigh, S. D.; Levenson, M.; Vangel, M.; Banks, D. L.; Heckert, N. A.; Dray, J. F. ; Vo, S. **Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications**. Technical report, Gaithersburg, MD, United States, 2010.

Boosten, M. **Transmission overhead and optimal packet size**. Disponível em: http://hsi.web.cern.ch/HSI/dshs/publications/wotug21/dsnic/html/node9.html, 1998. Acesso Outubro, 2017.

Bouchkaren, S.; Lazaar, S. A fast cryptosystem using reversible cellular automata. **International Journal of Advanced Computer Science and Applications (IJACSA)**, v.5, 2014.

Brady, R. Optimization strategies gleaned from biological evolution. **Nature**, v.317, n.6040, p. 804–806, 1985.

Brito, A. S.; Soares, S. S. R. F. ; Villela, S. M. **Criptografia de fluxo por autômatos celulares: uma abordagem baseada na combinação de busca tabu com simulated annealing**. In: ENIA – VI Encontro Nacional de Inteligência Artificial, 2017.

CIMM. **Recozimento**. Disponível em: http://www.cimm.com.br/portal/verbetes/exibir/521-recozimento, 2016. Acesso Outubro, 2016.

Callen, H. B. **Thermodynamics and an Introduction to Thermostatistics**. Wiley, 1985.

Tech, C. **Relembre os maiores vazamentos de informação**. Disponível em: https://canaltech.com.br/seguranca/relembre-os-maiores-vazamentos-de-informacao-de-2013-17910/, 2014. Acesso Outubro, 2017.

Daemen, J.; Rijmen, V. **The Design of RijndaeL: AES - The Advanced Encryption Standard (Information Security and Cryptography)**. Springer, 2002.

Mole, V. L. D.; Martins, J. G. ; Júnior, J. M. **Caixeiro viajante–uma abordagem baseada no algoritmo simulated annealing**, 2004.

Delyra, J. L. **Transformadas de Fourier - Série Métodos Matemáticos Para Física e Engenharia**, volume 2. Livraria da Física, 2014.

Diwakar, M.; Lal, N. ; Sharma, P. Text security using 2d cellular automata rules. 2013.

Dorigo, M.; Caro, G. D. **The ant colony optimization meta-heuristic**. In: Corne, D.; Dorigo, M.; Glover, F.; Dasgupta, D.; Moscato, P.; Poli, R. ; Price, K. V., editors, New Ideas in Optimization, p. 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

Dorigo, M.; Stützle, T. **Ant Colony Optimization**. Scituate, MA, USA: Bradford Company, 2004.

Dorigo, M.; Stützle, T. Ant colony optimization: overview and recent advances. **Techreport, IRIDIA, Universite Libre de Bruxelles**, 2009.

Feo, T. A.; Resende, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. **Oper. Res. Lett.**, v.8, n.2, p. 67–71, Abr. 1989.

Glover, F. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research**, v.13, n.5, p. 533–549, Mai 1986.

Goldbarg, E.; Goldbarg, M. ; Luna, H. **Otimização Combinatória e Metaheurísticas: Algoritmos e Aplicações**. Elvevier, 2015.

Gonzalez, R.; Woods, R. **Processamento de imagens digitais**. Edgard Blucher, 2000.

Islam, M. F.; Ali, M. M. ; Majlis, B. Y. Fpga implementation of an lfsr based pseudorandom pattern generator for mems testing. **International Journal of Computer Applications**, v.75, n.11, 2013.

Janela, M. **Fastest touch-screen text message record officially broken with fleksy keyboard**. Disponível em: http://www.guinnessworldrecords.com/news/2014/5/fastest-touch-screen-text-message-record-officially-broken-with-fleksy-keyboard-57380/, 2014. Acesso Outubro, 2017.

Kirkpatrick, S.; Gelatt, C. D. ; Vecchi, M. P. Optimization by simulated annealing. **SCIENCE**, v.220, n.4598, p. 671–680, 1983.

Krischer, T. C.; Weber, R. F. **Um estudo sobre a máquina enigma**, 2012.

Lenstra, J. K.; Shmoys, D.; Applegate, D. L.; Bixby, R. E.; Chvátal, V. ; Cook, W. J. **The traveling salesman problem: A computational study**, 2009.

Menezes, A. J.; van Oorschot, P. C. ; Vanstone, S. A. **Handbook of Applied Cryptography (Discrete Mathematics and Its Applications)**. CRC Press, 1996.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H. ; Teller, E. Equation of state calculations by fast computing machines. **The Journal of Chemical Physics**, v.21, n.6, p. 1087–1092, 1953.

Miller, F. **Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams**. C.M. Cornwell, 1882.

Mladenović, N.; Hansen, P. Variable neighborhood search. **Comput. Oper. Res.**, v.24, n.11, p. 1097–1100, Nov. 1997.

Osman, I.; Laporte, G. Metaheuristics: A bibliography. **Annals of Operations Research**, v.63, n.5, p. 511–623, Out. 1996.

Paar, C.; Pelzl, J. **Understanding Cryptography: A Textbook for Students and Practitioners**. Springer Berlin Heidelberg, 2009.

Pavão, R. **Teoria da informação**. Disponível em: http://www.ib.usp.br/~rpavao/Teoria_da_Informacao.pdf, 2011. Acesso Dezembro, 2017.

Poorghanad, A.; Sadr, A. ; Kashanipour, A. **Generating high quality pseudo random number using evolutionary methods**. In: Computational Intelligence and Security, 2008. CIS'08. International Conference on, volume 1, p. 331–335. IEEE, 2008.

Raggo, M. T.; Hosmer, C. **Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems, Mobile Devices and Network Protocols**. 1st. ed., Syngress Publishing, 2013.

Rivest, R. L.; Shamir, A. ; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. **Commun. ACM**, v.21, n.2, p. 120–126, Fev. 1978.

Seredynski, M.; Bouvry, P. **Block encryption using reversible cellular automata**. In: International Conference on Cellular Automata, p. 785–792. Springer, 2004.

Shannon, C. A mathematical theory of communication. **Bell System Technical Journal**, v.27, p. 379–423, 623–656, 1948.

Singh, S. **The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography**. Anchor, 2011.

de Souza, S. A. F. S. **The margolus neighbourhood**, 2001. Monografia (Licenciatura em Ciencia de Computadores), Faculdade de Ciências da Universidade do Porto, Portugal.

Stützle, T.; Dorigo, M. Aco algorithms for the traveling salesman problem. **Evolutionary Algorithms in Engineering and Computer Science**, p. 163–183, 1999.

Tomassini, M.; Perrenou, M. **Cryptography with cellular automata**. In: Applied Soft Computing 1, p. 151–160, 2001.

Tyler, T. **The margolus neighbourhood**. Disponível em: http://cell-auto.com/neighbourhood/margolus/, 2016. Acesso Junho, 2016.

Villela, S. M.; Carvalho, L. A. V. **O resfriamento simulado no projeto ótimo de autômatos celulares para a geração de chaves em criptografia de fluxo**. In: ENIA – VI Encontro Nacional de Inteligência Artificial, p. 1082–1091, 2007.

Wolfram, S. Cellular automata as simple self-organizing systems. **S. Caltech Preprint CALT-68-938**, 1982.

Wolfram, S. Cellular automata as models of complexity. **Nature**, v.311, n.5985, p. 419–424, 1984.

Wolfram, S. **Cryptography with cellular automata**. In: Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings, p. 429–432, 1985.

Wolfram, S. **Cryptography with cellular automata**. In: Advances in Cryptology, CRYPTO '85, p. 429–432, London, UK, UK, 1986. Springer-Verlag.

Wolfram, S. Random sequence generation by cellular automata. **Advances in applied mathematics**, v.7, n.2, p. 123–169, 1986.

Wolfram, S. **A new kind of science**. Wolfram-Media, 2002.

Wykes, S. **Criptografia Essencial (Em Portuguese do Brasil)**. Elsevier, 2016.

Zeng, K.; Yang, C. H.; Wei, D. Y. ; Rao, T. R. N. Pseudorandom bit generators in stream-cipher cryptography. **Computer**, v.24, n.2, p. 8–17, Fev. 1991.

Zuse, K. Rechnender raum (calculating space). **Schriften Zur Dataverarbeitung**, v.1, 1969.

A Resultados Completos

Este apêndice contém as tabelas com os resultados completos do primeiro experimento e do experimento comparativo com outras abordagens presentes na literatura apresentados no Capítulo 5. Essas tabelas foram geradas com o intuito de permitir um melhor entendimento dos resultados obtidos por meio dos experimentos executados.

Tabela A.1: AC 8 Bits - Entropia

Seed	SA1	SA1+R	SA1+R+Tabu	ACO	SA2	SA2+R	SA2+R+Tabu
475	7,9922	7,9922	7,9922	5,3783	7,9922	7,9922	7,9922
726	7,9922	7,9922	7,1797	5,9889	7,9922	7,9922	7,9922
979	6,0439	7,9922	7,9922	5,4558	5,8894	7,9922	7,9375
2571	7,1641	7,9922	7,1641	5,3783	7,1641	7,9922	7,1641
2643	6,0887	6,0887	7,9922	5,5160	6,0887	6,0887	7,9922
2900	7,9766	7,7188	7,9766	5,5011	7,7344	7,9922	7,9609
3128	5,7968	7,9922	6,7199	5,7485	5,7969	7,9922	7,9922
3208	5,8522	5,8845	7,7578	5,5780	7,9375	5,9448	7,9922
3314	6,1544	5,8865	7,9688	5,6251	6,0575	5,8865	7,9922
3750	7,9922	7,9922	7,9922	5,5164	7,9922	7,9922	7,9922
4367	7,9922	7,9922	7,9844	5,7264	7,9922	7,7188	7,9922
4962	7,8203	7,9378	7,5313	5,7013	7,9375	7,9922	7,9922
5609	7,9922	7,9922	7,9922	5,8129	6,0127	6,0127	7,9922
5936	7,9922	7,9922	7,9922	5,8411	7,9922	7,9922	7,9922
6411	7,9922	5,8724	7,9922	5,4714	7,9922	7,9922	7,9922
6600	7,9688	7,9688	7,8750	5,4393	7,9922	7,9922	7,9922
6873	7,9922	5,9749	7,9922	5,6251	7,9922	7,9844	7,9922
7245	7,9922	7,9922	7,9922	5,1857	7,9922	7,9922	7,9922
8582	7,9922	6,1362	7,9922	5,5439	7,9922	6,1362	6,5954
8771	7,7578	5,7123	7,0469	5,6099	7,9922	5,7123	7,1563
9258	7,9922	6,8019	7,9922	5,3659	7,9922	6,8019	7,9922
13858	7,9922	7,9922	7,9922	5,4186	7,9922	7,9922	7,9375
17951	6,0195	7,9922	7,0703	5,3339	5,7166	7,9922	7,4375
38222	5,987'	7,0625	7,9922	5,5476	5,9871	7,0625	7,9922
42614	7,9922	7,9922	7,9922	5,5327	7,9922	7,9922	7,9922
56895	7,9922	7,9922	7,4063	6,1557	7,9922	7,9922	7,9768
69320	6,6710	7,9922	7,9922	5,4148	7,9922	7,9922	7,9922
76359	7,9922	7,9922	7,9766	5,5479	7,9922	7,9922	7,9922
86707	5,9369	6,3602	7,9922	5,4393	7,7500	5,9196	6,6807
96808	7,9922	7,9922	7,9922	5,4714	7,9766	7,9922	7,4532

Tabela A.2: AC 12 Bits - Entropia

Seed	SA1	SA1+R	SA1+R+Tabu	SA2	SA2+R	SA2+R+Tabu
475	11,9619	11,9619	11,9619	11,9624	11,9995	11,8672
726	10,8594	10,9471	11,9380	10,8587	11,9902	11,5422
979	11,6304	11,2412	11,2305	11,9441	11,9375	11,9161
2571	10,9193	11,9648	10,9193	10,9193	11,9996	11,7861
2643	10,6380	11,9604	10,6374	11,7710	11,9766	11,9990
2900	11,8457	11,8457	11,8457	11,9380	11,9990	11,9336
3128	11,8652	11,9341	11,9775	11,3540	6,5437	10,8210
3208	11,1787	11,9570	11,8530	11,7725	11,9961	11,9995
3314	7,2927	7,0238	11,9766	7,29274	11,2676	10,5249
3750	10,9424	10,3035	11,9995	10,9424	10,9726	11,7617
4367	11,9570	11,9727	11,9995	11,9570	7,0829	11,9287
4962	11,7285	11,9980	11,3184	11,9976	11,7144	11,9922
5609	11,6748	11,9893	11,9922	11,3926	11,0596	11,8164
5936	11,8672	11,9990	11,8672	10,1466	11,9995	11,9995
6411	11,9990	11,9995	11,9380	11,9990	11,9995	11,9478
6600	11,6357	11,8994	11,6357	10,6860	11,9616	11,7871
6873	11,0225	11,9961	11,9194	10,3972	11,4780	11,0405
7245	11,9990	7,1307	10,7524	10,7882	10,7318	11,9995
8582	10,8673	11,9990	11,9521	11,1504	11,9907	11,5894
8771	11,9990	11,9346	11,9990	11,9995	10,2026	11,9907
9258	11,6680	11,9346	11,6680	11,9741	9,10229	11,9932
13858	11,9980	11,9980	11,7227	11,6680	11,6816	11,6680
17951	11,9883	11,4380	11,9883	11,5078	11,5996	11,9922
38222	11,0488	11,1655	11,1689	11,1694	11,9995	11,9990
42614	11,4077	6,9558	11,4077	11,4883	7,1758	11,9961
56895	11,9927	11,4478	11,9929	11,9961	11,9985	11,9126
69320	11,1162	11,9741	11,4956	11,9375	6,9116	11,9224
76359	11,7163	11,8167	11,4717	11,6958	11,7178	11,9736
86707	11,9590	11,4880	11,9590	11,9332	6,9803	11,9580
96808	11,0330	11,8467	11,2729	11,9380	11,9341	11,9746

Tabela A.3: AC 16 Bits - Entropia

Seed	SA1	SA1+R	SA1+R+Tabu	SA2	SA2+R	SA2+R+Tabu	Thomassini	Rule30
475	14,2755	14,0271	15,9543	15,9808	15,0785	15,2213	6,9773	12,5640
726	15,9566	9,3027	15,9261	15,7313	9,3021	15,4710	5,9069	12,0170
979	14,8232	15,0615	15,9959	14,7568	15,9424	14,7087	7,4094	12,0357
2571	15,1145	14,4291	15,9072	14,4733	15,0490	15,4569	7,9887	12,0286
2643	15,4934	15,5520	15,9534	15,3951	14,7395	15,8507	6,9776	12,5541
2900	15,9999	15,8164	15,0233	14,6703	14,6773	15,7415	5,9071	5,3256
3128	15,9781	15,8365	15,6556	15,9573	14,7299	15,9766	6,1293	12,5549
3208	15,9999	15,7880	15,4560	13,8018	15,7313	15,7483	8,9943	12,5586
3314	15,8653	14,1242	15,9860	15,8713	15,9964	14,3829	6,6724	12,0191
3750	14,7893	14,3808	15,0605	15,0575	15,8022	14,7689	6,9542	12,0253
4367	15,9280	15,9828	15,9226	14,5092	15,3616	15,9121	7,9888	12,5582
4962	14,3007	15,7131	15,2074	14,9079	15,8188	15,8713	6,1295	12,5553
5609	14,5835	15,3954	13,9339	14,4470	15,9549	15,4708	5,3932	12,0267
5936	15,3997	15,6423	15,0132	15,0785	14,6623	14,6028	6,9778	12,5682
6411	13,2355	15,7363	15,9487	15,8146	15,9999	14,6863	7,7142	12,5542
6600	15,7821	14,3765	15,6833	14,6167	15,2790	14,7499	7,9887	12,0304
6873	14,9658	15,8293	15,1844	13,9980	15,9363	15,4008	7,9773	12,0164
7245	14,5319	15,6086	14,4234	15,1092	15,3099	15,4172	4,9069	12,0254
8582	15,9146	14,5466	15,1794	14,0288	15,5912	15,1748	3,3225	12,0198
8771	15,7313	13,6737	15,2615	12,9298	12,6502	15,9884	7,4094	12,5622
9258	14,2223	12,9272	14,5936	12,9272	13,2056	15,9379	4,1701	12,0188
13858	13,8722	9,26201	15,9299	15,2061	15,7279	15,9465	6,9774	12,0230
17951	14,0084	14,9325	15,8616	15,9534	14,9604	15,8505	6,9774	12,5540
38222	12,7382	14,1527	15,9072	15,3074	14,2608	15,9919	6,9774	12,0179
42614	14,4966	14,3472	15,9611	14,6654	15,7312	15,9844	6,9071	12,0281
56895	13,4930	14,5521	15,9999	15,9366	12,2732	14,8140	5,9076	12,5611
69320	14,4766	13,7632	15,6688	14,6144	15,7550	15,7889	7,9887	12,5667
76359	13,1864	15,9224	15,0295	15,3048	15,9840	15,1783	8,7616	12,0178
86707	15,9309	14,7793	14,8871	15,3481	14,9112	15,8860	5,9069	3,0104
96808	15,1926	15,1324	15,7615	13,3390	15,7298	14,6798	4,9075	12,0319

Tabela A.4: Tempos de 5 execuções do SA1+R+Tabu (em segundos)

Seed	**Tempo**
33	3036,01
42	3761,30
574	3186,57
767	3624,85
925	3290,18

B NIST

Neste apêndice estão as informações da bateria de teste do NIST utilizada.

Tabela B.1: Testes estatísticos do NIST

Teste	Descrição
Frequência	Determina a proporção de zeros e uns em uma sequência. Para ser aleatória, deve ter praticamente o mesmo número de zeros e uns
Frequência em Blocos	Determina a proporção de uns dentro de blocos de bits de tamanho definido M, no caso específico $M = 16$
Corridas	Verifica o número de *corridas* em uma sequência, onde uma *corrida* é uma sequência ininterrupta de bits idênticos
Corrida Mais Longa	Determina se o tamanho da maior *corrida* é consistente com o valor esperado em uma sequência totalmente aleatória
Posto de Matriz Binária	Verifica a dependência linear entre subsequências de tamanho fixo da sequência original
Transformada Discreta de Fourier	Analisa a altura dos picos da Transformada Discreta de Fourier da sequência, identificando estruturas periódicas (Delyra, 2014), isto é, padrões iguais muito próximos uns dos outros
Não Sobreposição de Padrão	Detecta geradores que produzem muitas vezes um mesmo padrão não periódico, verificando o número de *strings* alvos pré-definidas
Sobreposição de Padrão	Semelhante ao teste de Não Sobreposição de Padrão, porém com pequenas mudanças no funcionamento interno
Estatística Universal de Maurer	Verifica se uma sequência pode ou não passar por uma compressão significativa sem perda de informação. Uma sequência com essas características é considerada não aleatória
Complexidade Linear	Verifica se uma sequência é complexa o suficiente para ser considerada randômica, analisando o tamanho dos Registradores de Deslocamento com Retroalimentação Linear (Islam et al, 2013) gerados a partir da sequência, quanto maior o valor, mais complexidade
Serial	Determina a frequência de todas as possíveis subsequências de um tamanho definido dentro da sequência. Uma sequência aleatória tem a mesma frequência para todas as suas subsequências
Entropia Aproximada	Calculado da mesma forma descrita na subseção 2.3
Somas Cumulativas	Determina se a soma acumulativa das sequências parciais é igual ao valor esperado em sequências aleatórias, que é próximo de 0
Excursões Aleatórias	Determina se o número de visitas a um estado particular em um ciclo desvia do valor esperado para uma sequência aleatória dentro de um processo de soma cumulativa
Variante de Excursões Aleatórias	Verifica o número de vezes que um estado em particular é visitado em um processo de soma cumulativa sobre a sequência

Tabela B.2: Parâmetros NIST (*Valores recomendados pelo NIST)

Parâmetro	**Valor**
Teste de Frequência em Bloco - Tamanho do Bloco	16
Teste de Sobreposição de Padrão - Tamanho do Bloco	16
Teste de Não Sobreposição de Padrão - Tamanho do Bloco	16
Teste de Entropia Aproximada - Tamanho do Bloco	10*
Teste Serial - Tamanho do Bloco	16
Teste de Complexidade Linear - Tamanho do Bloco	500*