O Resfriamento Simulado no Projeto Ótimo de Autômatos Celulares para a Geração de Chaves em Criptografia de Fluxo

Saulo Moraes Villela[1], Luís Alfredo Vidal de Carvalho[1]

[1]COPPE – Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 68.511 – 21.941-972 – Rio de Janeiro – RJ – Brazil
{saulomv,alfredo}@cos.ufrj.br

Abstract. *Cryptography has become a basic requirement in this age of global electronic connectivity to secure data storage and transmission against the possibility of message eavesdropping and electronic fraud. Cellular automata have been studied as cryptography algorithms. The present work presents a cellular automata model for the generation of keys to flow cryptography with the use o the Simulated Annealing heuristics. Exhaustive tests have shown interesting results, mainly the difficulty of finding good keys. Our experiments deny previous results reported in the literature which establish rules for good key cellular automata generators.*

Resumo. *A criptografia se tornou um requisito básico nessa era de conectividade eletrônica global para assegurar o armazenamento de dados e transmissões contra a possibilidade de interceptação de mensagens e fraudes eletrônicas. Autômatos celulares têm sido estudados como uma opção de técnica de criptografia. No presente trabalho, utilizamos autômatos celulares para gerar chaves para uma criptografia de fluxo. Para isso, fizemos o uso da meta-heurística Resfriamento Simulado. Extensivos testes nos mostraram diversos resultados e, com eles, conseguimos observar a enorme dificuldade que se é encontrar boas chaves. Combatemos, com isso, os resultados obtidos por trabalhos anteriores.*

1. Introdução

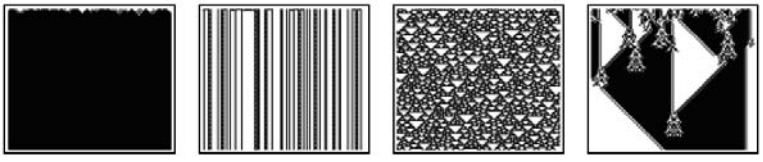
Hoje em dia, as redes globais são caracterizadas pelo enorme crescimento da necessidade do armazenamento e transmissão de informações digitais. Com isso, a grande maioria das organizações tem se tornado crescentemente dependente de técnicas criptográficas para garantir a segurança e autenticidade em diversas áreas. A criptografia tem uma longa história e diversas técnicas criptográficas diferentes já foram sugeridas. Aqui, iremos descrever a aplicação de alguns tipos de autômatos celulares neste domínio.

Autômatos celulares (ACs) foram usados como mecanismos de encriptação por Wolfram (1986) e Nandi et al. (1994). Outros autores usaram ACs para criptografia de chave-pública, mas seus trabalhos não serão discutidos aqui, uma vez que usamos ACs para sistemas simétricos, onde as chaves de encriptação e desencriptação são a mesma ou podem ser calculadas uma a partir da outra. Nosso esquema de encriptação é baseado na geração de seqüências de bits pseudo-aleatórios por autômatos celulares.

2. Autômatos Celulares

Autômatos celulares são sistemas dinâmicos nos quais espaço e tempo são discretos. Um autômato celular consiste de um vetor de células, cada uma podendo ser em um de um finito número se possíveis estados, atualizado sincronamente em passos de tempo discretos, de acordo com uma regra interativa local. Aqui, nós iremos somente considerar autômatos Booleanos, para cada estado da célula $s \in \{0,1\}$. O estado de uma célula no passo seguinte de tempo é determinado pelos estados atuais de uma vizinhança acerca das células. O vetor celular é *d*-dimensional, onde $d = 1, 2, 3$ é usado na prática; no nosso trabalho, nós concentraremos em $d = 1$, isto é, vetores unidimensionais. A regra contida em cada célula é essencialmente uma máquina de estado finito, usualmente especificada na forma de uma tabela de regras (também conhecida como função de transição), com uma entrada para cada possível configuração de vizinhança de estados. A vizinhança celular de uma célula consiste no próprio estado e nos estados das células adjacentes. Para ACs unidimensionais, uma célula é conectada a *r* vizinhos locais (células) em cada lado onde *r* é referente ao raio (então, cada célula tem uma vizinhança (δ) de $2r + 1$ vizinhos). Quando consideramos o reticulado de tamanho finito, condições de contorno periódico são freqüentemente aplicadas, resultando em um reticulado circular no caso unidimensional. Autômatos celulares não-uniformes (também conhecidos como não-homogêneos) são iguais aos uniformes com a exceção que as regras não precisam ser idênticas para todas as células.

Um dos pontos mais importantes dos autômatos celulares é a classificação de suas regras de atualização *f*, e, conseqüentemente, dos próprios autômatos. O interesse nesta questão recebeu sua atenção inicial a partir do estudo dos autômatos celulares infinitos unidimensionais por Stephen Wolfram (2002), que resultaram na descoberta empírica de que, desconsiderando o estado inicial do autômato, *f* consistentemente recai dentro de uma das quatro possíveis categorias qualitativas, como mostra a Figura 1: (i) a evolução do autômato leva a uma configuração homogênea; (ii) a evolução leva a um ponto fixo não-homogêneo ou a um ciclo; (iii) a evolução leva a uma sucessão caótica de configurações; ou (iv) a evolução leva a estruturas espaço-temporais localizadas que são algumas vezes “persistentes”.



(i) (ii) (iii) (iv)

Figura 1. As figuras (i)–(iv) mostram a evolução de autômatos celulares com quatro regras de atualização diferentes a partir de estados iniciais aleatórios. Nesses casos, temos dois estados possíveis $S = \{0, 1\}$ e vizinhança $\delta = 3$.

3. Criptografia de Fluxo

Algoritmos de fluxo (*stream chiphers*) (Figura 2) são algoritmos de chave simétrica que operam sobre cada *bit* da mensagem individualmente. Estes algoritmos são úteis quando o transmissor da mensagem precisa enviar para o receptor um fluxo contínuo de dados criptografados, como, por exemplo, um computador que está permanentemente transmitindo dados criptografados para outro computador.

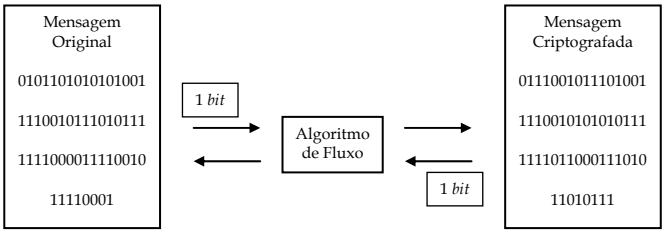


Figura 2. Algoritmos de fluxo.

Nos algoritmos de fluxo, uma operação matemática (L) precisa ser realizada com cada *bit* da mensagem separadamente de forma a alterar-lhe o valor, criptografando-o. A operação L é normalmente uma função "ou-exclusivo". É suficiente para isto que convencionemos que um dos *bits* a serem operados por L pertença à mensagem original a ser encriptada enquanto o outro *bit* necessário à operação seja uma chave de encriptação. O resultado da operação L sobre o *bit* da mensagem e sobre a chave de encriptação é o terceiro *bit* que representa um *bit* da mensagem criptografada.

A chave de encriptação deve ser constituída de *bits* que variam no tempo, encriptando cada *bit* da mensagem original diferentemente. O ideal seria que a chave de encriptação fosse uma seqüência de *bits* aleatoriamente gerados e, portanto, imprevisível para qualquer intruso desejoso de criptanalisar a mensagem. A chave de encriptação deve possuir o mesmo número de *bits* da mensagem original, pois, só assim, cada *bit* da mensagem poderá ser encriptado por um *bit* da chave diferente. A chave de encriptação é uma seqüência aleatória de n *bits*, na qual n é o número de *bits* da mensagem original. Chama-se esta seqüência de *bits* de chave de fluxo. Basicamente, todos os algoritmos de fluxo possuem a estrutura indicada na Figura 3a. O processo de desencriptação é semelhante ao de encriptação, bastando utilizar-se a operação matemática inversa L_{inv} sobre a mensagem criptografada e com a mesma chave de fluxo (Figura 3b).

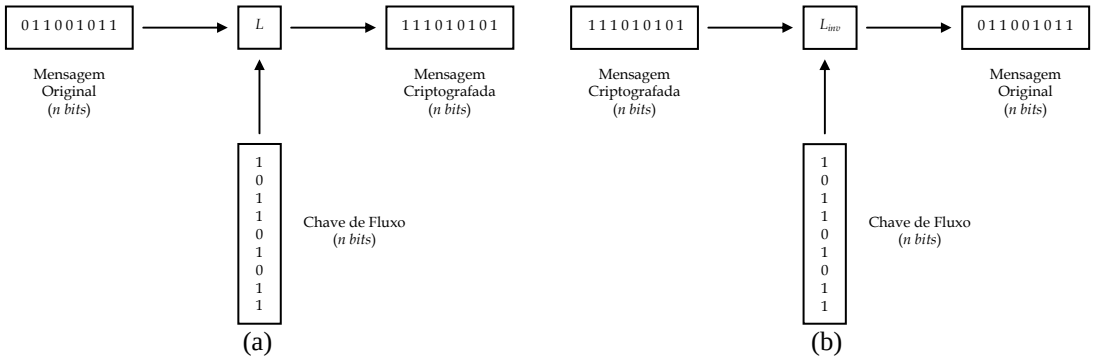


Figura 3. (a) Encriptação e (b) desencriptação dos algoritmos de fluxo.

Como na desencriptação a chave de fluxo utilizada deve ser a mesma usada no processo de encriptação, um problema conceitual e operacional surge: se a chave de fluxo da encriptação for realmente aleatória, ela nunca poderá ser gerada novamente para o processo de desencriptação. Isso uma vez que assumimos que a chave de fluxo não é guardada ou transmitida para o receptor, pois seria muito prejudicial à segurança do algoritmo. A chave de fluxo deve, então, ser gerada no ato de encriptação e, depois, reproduzida, na desencriptação. Daí, a chave de fluxo não poder ser de fato aleatória, mas sim, uma seqüência de *bits* denominada de pseudo-aleatória, isto é, uma seqüência que possui propriedades de aleatoriedade, mas pode ser reconstruída quando se desejar.

Os algoritmos de fluxo necessitam de alguma máquina, ou processo, gerador de chaves de fluxo para poderem funcionar. Este gerador de chaves de fluxo deve ser capaz de produzir, para a encriptação, e reproduzir, para a desencriptação, a mesma chave de fluxo com características de aleatoriedade suficientes para garantir a segurança do algoritmo. Os geradores de chave de fluxo, em geral, possuem uma chave-semente que dispara o processo de geração de uma chave de fluxo específica, dentre todas as possíveis de serem geradas pelo mesmo. Quando o transmissor deseja criptografar uma mensagem, ele escolhe uma chave-semente e encripta a mensagem com a seqüência pseudo-aleatória de *bits* (chave de fluxo) produzida pelo gerador de chaves de fluxo. O receptor da mensagem deve conhecer a chave-semente escolhida pelo transmissor, pois só assim poderá iniciar o processo de geração da mesma chave de fluxo e desencriptar a mensagem. A chave-semente tem o papel de determinar os parâmetros internos do gerador de chaves de fluxo, especificando o seu funcionamento e obrigando-o a gerar uma chave de fluxo particular. No interior do gerador de chaves de fluxo encontramos um registrador capaz de armazenar um certo número de *bits*, denominado estado interno. Com base neste estado interno, um outro módulo do gerador, módulo de saída, gera um *bit* da chave de fluxo. Finalmente, um terceiro elemento do gerador de chaves de fluxo, o módulo de atualização, atualiza o estado interno, alterando-lhe o valor para que um novo *bit* da chave de fluxo possa ser produzido pelo módulo de saída, futuramente. A Figura 4 ilustra o interior de um gerador de chaves de fluxo.

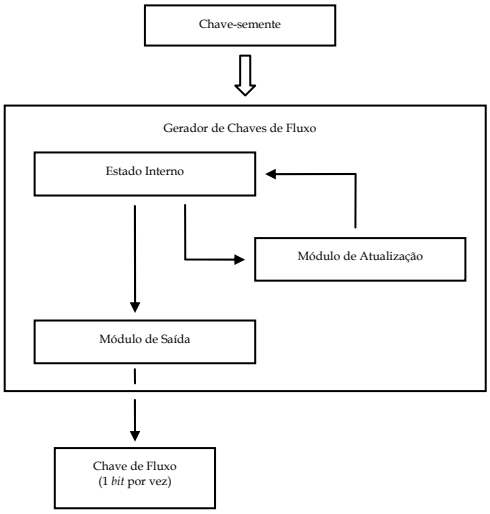


Figura 4. Gerador de chaves de fluxo

4. Resfriamento Simulado

O Resfriamento Simulado é uma meta-heurística de busca onde uma solução experimental é modificada aleatoriamente. Uma "energia" é definida para representar quão boa é a solução. O objetivo é encontrar a melhor solução maximizando esta energia. Mudanças que conduzem a uma energia mais alta são aceitas imediatamente; um decréscimo é aceito probabilisticamente. A probabilidade p é dada por $p = e^{-df/KT}$, onde df é a variação "energética", K é a constante de Boltzmann e T é a "temperatura". Inicialmente a temperatura é elevada, correspondendo a um líquido ou estado derretido onde as mudanças grandes são possíveis, e ela é reduzida progressivamente usando um resfriamento que permite assim mudanças menores até que o sistema se solidifica em uma solução alta da energia.

Sendo D um conjunto finito, f uma função matemática dita função objetivo e $D \subseteq D^n$ o conjunto das chamadas soluções viáveis, estabelecemos o método do Resfriamento Simulado por:

$$\max f(x)$$

$$\text{sujeito a } x \in D,$$

Seja $x_0 \in D$ uma solução viável inicial,
T_0 uma temperatura inicial e r uma taxa de decréscimo da temperatura;
$k := 0$; $x_k := x_0$; $T_k := T_0$;
enquanto o sistema não está "cristalizado" **faça**
início
enquanto o sistema não atinge o "equilíbrio térmico" à temperatura T_k **faça**
início
selecione aleatoriamente uma solução viável $x' \in D$;
$\Delta f := f(x') - f(x_k)$;
se $\Delta f \geq 0$ **então** $x_k := x'$;
do contrario $x_k := x'$ com probabilidade $p = e^{-\Delta f / T_k}$;
fim;
$T_{k+1} := r(T_k)$; $x_{k+1} := x_k$; $k := k + 1$;
fim;

---

a solução é x_k.

Para a determinação completa do algoritmo é preciso definir quando o sistema está "cristalizado" e atinge o "equilíbrio térmico". O sistema atinge o "equilíbrio térmico" quando a média aproximada da função objetivo para uma dada temperatura estabiliza-se, sofrendo apenas pequenas flutuações ao redor de seu valor médio exato e conhecido. Já a "cristalização" do sistema ocorre quando, ao reduzir-se a temperatura, a média da função objetivo não aumenta, significando, na melhor hipótese, que o sistema atingiu seu máximo global (Figura 5). A taxa de decréscimo da temperatura deve ser suficientemente pequena para garantir a "cristalização" do sistema na configuração do máximo global, porém não tão baixa que inviabilize a aplicação do método por requerer um número extremamente grande de configurações simuladas.

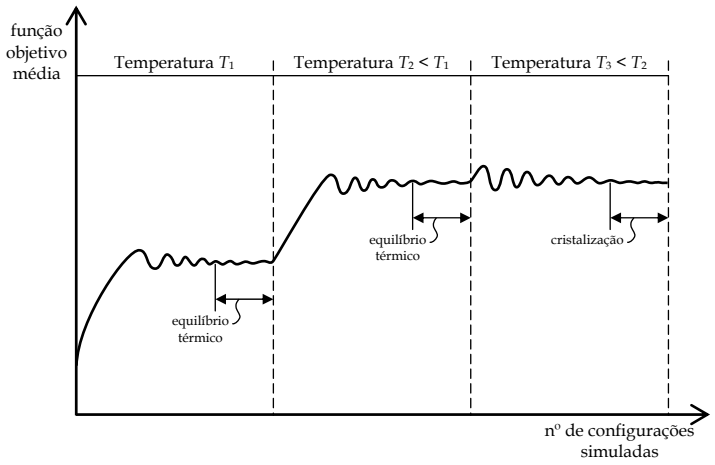


Figura 5. Equilíbrio térmico e cristalização no Resfriamento Simulado.

4.1. Entropia

Para medir a homogeneidade de determinados grupos, alguns algoritmos se utilizam do conceito de variância ou de entropia. O conceito de entropia está associado à desordem. Nas leis da Termodinâmica encontramos a afirmação de que qualquer máquina que gere trabalho a partir de calor, só pode funcionar se houver uma fonte de temperatura mais alta e uma fonte de temperatura mais baixa às quais a máquina esteja conectada. Em outras palavras, sempre é preciso uma diferença de temperaturas entre as quais possa fluir energia (calor) de uma fonte quente para uma fonte fria, sendo parte desta energia em fluxo transformada no trabalho útil realizado pela máquina. Entropia é uma medida de diversidade, degeneração, desordem, desorganização e caos.

Definimos a entropia da informação como a quantidade média de informação necessária para entendermos um fenômeno. Se um fenômeno depende de n fatores ou eventos E_i, onde $P(E_i) = p_i$, $i = 1, 2, 3, \ldots, n$, a entropia da informação sobre o fenômeno é a média da quantidade de informação que necessitamos para prever cada evento:

$$H = \sum_{i=1}^{n} p_i \log_2 p_i$$

onde p_i é a probabilidade de ocorrência do evento E_i.

Se um fenômeno complexo e desorganizado depende de vários eventos de difícil previsão, a quantidade de informação para prevermos cada evento é alta e, certamente, a média destas quantidades também será alta, resultando em uma grande entropia da informação. Por outro lado, um fenômeno que dependa de muitos eventos facilmente previsíveis, terá uma entropia de informação baixa, pois a média de baixas quantidades de informação será um número pequeno.

Se na Termodinâmica a entropia é a perda presente de trabalho útil irrecuperável no futuro, na Ciência da Computação, a entropia é a falta de conhecimento no presente que deve ser suprida no futuro. Na Termodinâmica, a entropia é a desordem por degradação enquanto na Ciência da Computação a entropia é a desordem por falta de conhecimento. É justamente essa imprevisibilidade que necessitaremos para gerar a chave de cifra utilizada em nosso sistema. A entropia do autômato celular será utilizada como função objetivo do algoritmo do Resfriamento Simulado.

5. O Método Proposto

5.1. O Método

O método que propomos foi a geração de chaves de criptografia de fluxo a partir da seleção de autômatos celulares “ótimos” escolhidos através da aplicação da meta-heurística do Resfriamento Simulado. O método combate resultados de outros trabalhos.

Os testes foram realizados com autômatos celulares unidimensionais de 8 *bits* e vizinhança de tamanho $\delta = 3$, ou seja, os vizinhos de uma célula considerados são a célula à sua esquerda, ela própria e a célula à sua direita. Os autômatos utilizados são não-uniformes, isto é, a regra de cada célula não necessariamente é idêntica à das outras. Com isso, para cada simulação temos 8 regras distintas, uma para cada célula, fazendo, assim, com que a solução possa se tornar ainda mais aleatória.

Como foi mencionado na seção anterior, utilizamos o valor da entropia para ser a função objetivo do Resfriamento Simulado, uma vez que entropia é desordem, falta de conhecimento, ou seja, uma excelente forma de determinarmos se o autômato gerado segue ou não algum "padrão". A entropia foi calculada gerando-se 1024 passos do autômato celular, de acordo com suas regras de transição. O cálculo da entropia E_h para o conjunto de k^h possíveis subseqüências de tamanho h é dado por:

$$E_h = -\sum_{j=1}^{k^h} ph_j \log_2 ph_j$$

onde h_1, h_2, ..., h_{k^h} são todas as possíveis subseqüências de tamanho h. O algoritmo utilizado foi o seguinte:

1. *Gerar 8 bits aleatórios (estado inicial do autômato) e oito regras aleatórias de oito bits (entre 0 e 255) para este autômato;*
2. *A partir desta configuração inicial, escolher uma "temperatura" inicial alta e determinar um número mínimo de iterações por "temperatura";*
3. *Gerar 1024 passos do autômato, de acordo com as regras de transição, e calcular a entropia;*
4. *Selecionar, aleatoriamente, uma das 8 regras do autômato e um novo valor para ela, também de 8 bits (entre 0 e 255), e calcular a entropia desse autômato, gerando 1024 passos dele;*
5. *Calcular a diferença Δ entre o valor da entropia do autômato com a regra sorteada alterada e o autômato com o valor da regra sorteada original:*
    a. *Se Δ for positivo, aceitar a nova configuração do autômato com a regra escolhida alterada como sendo o seu valor efetivo a partir deste instante;*
    b. *Se Δ for negativo, calcular a probabilidade $p = e^{-\Delta/KT}$, onde K é a constante de Boltzmann, de aceitar a nova configuração do autômato como possível. Isto é feito sorteando-se um número aleatório e verificando se ele cai entre zero e o valor p recentemente calculado:*
        i. *Se o número aleatório está dentro do intervalo zero e p, a nova configuração do autômato é aceita como o efetivo a partir de então;*
        ii. *Do contrário esta configuração é rejeitada e a regra sorteada retoma seu valor original de antes do sorteio;*
6. *Calcular a diferença das médias da entropia atual e da entropia há x (mínimo de iterações por temperatura) passos atrás, ou seja, verificar se o sistema atingiu o "equilíbrio térmico":*
    a. *Se a diferença for menor que um dado valor (parâmetro ε), diminuir a "temperatura" levemente (parâmetro α) e retornar ao passo 4;*
    b. *Senão, somente retornar ao passo 4, com o mesmo valor da "temperatura";*
7. *O término do processo ocorre quando a "temperatura" é próxima de zero (parâmetro temperatura mínima). A configuração do autômato com suas regras neste ponto será a solução do problema, o máximo global da função, ou seja, a entropia máxima do autômato.*

Utilizamos como verificação da "cristalização", para o término do processo, o parâmetro "temperatura" mínima, ou seja, independentemente se a média da entropia de uma "temperatura" foi ou não superior ou igual à média da entropia da "temperatura" anterior, o sistema somente pára quando se atinge essa "temperatura" mínima. Isso para que o simulador gere mais soluções ótimas, e não pare na primeira encontrada.

A determinação da "temperatura" inicial e dos outros parâmetros do problema é de suma importância para a obtenção de resultados ótimos. Exaustivos testes resultaram nos seguintes valores para esses parâmetros:

- *Temperatura inicial: 0,1;*
- *Decréscimo da temperatura (α): 0,95;*
- *Temperatura mínima: 0,001;*
- *Número mínimo de iterações por temperatura: 200;*
- *Diferença das médias (ε): 0,01.*

5.2. Resultados

Com esses parâmetros, em 65% dos casos, e num tempo médio de 1 hora, foram obtidas soluções consideradas ótimas. A Tabela 1 contém 16 exemplos dessas soluções.

Tabela 1. Exemplos de soluções ótimas (entropia superior a 7,8)

Regra 1	Regra 2	Regra 3	Regra 4	Regra 5	Regra 6	Regra 7	Regra 8	Entropia
85	86	90	170	101	170	170	106	**7,9937**
106	169	85	86	90	153	105	170	**7,9937**
169	165	89	153	154	150	170	85	**7,9937**
90	102	90	105	90	170	170	90	**7,9874**
105	105	105	90	105	120	51	153	**7,9874**
150	150	71	99	150	165	85	105	**7,9874**
89	240	51	105	90	150	30	57	**7,9748**
90	105	90	150	170	15	197	105	**7,9748**
165	150	90	150	170	15	197	105	**7,9748**
195	102	105	90	86	240	85	225	**7,9497**
150	150	45	99	106	165	85	105	**7,9434**
149	102	90	154	150	170	170	89	**7,9183**
150	105	75	54	170	15	85	105	**7,8994**
169	86	90	170	101	89	170	106	**7,8931**
90	102	90	106	90	149	170	90	**7,8805**
85	86	105	154	150	170	170	149	**7,8554**

A chave-semente S será formada, então, pelo vetor com as 8 regras de atualização R_i e o os 8 *bits* aleatórios do estado inicial do autômato $C(0)$, ou seja, $S = \langle R_i, C(0) \rangle$, $1 \le i \le 8$. Com isso, temos uma chave-semente pequena que, quando colocada no gerador de chave de fluxo, gera a chave K que tem 8192 *bits* e é pseudo-aleatória. Com os resultados obtidos, concluímos que não importa somente descobrirmos regras consideradas "boas", mas também em quais posições essas regras estarão. Para demonstrar essa necessidade, realizamos alguns testes com as soluções satisfatórias. Sorteamos as posições dos conjuntos de regras dessas soluções (Figura 6).

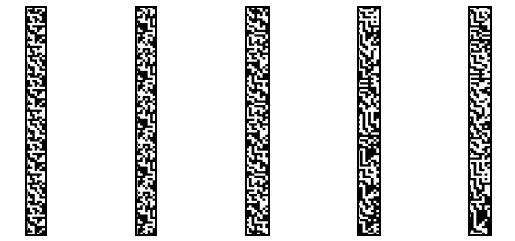


Figura 6. Regras 150, 85, 153, 105, 85, 150, 165 e 90 em posições aleatórias.

Observando essa enorme importância que a posição das regras tem, tentamos criar uma heurística com as posições mais freqüentes. Para isso criamos uma tabela com as regras obtidas, juntamente com suas posições, de 471 exemplos de soluções ótimas. De posse desta tabela, simulamos algumas combinações de regras e posições, tentando sempre pegar regras que aparecem com freqüência na posição presente na tabela. Os resultados podem ser vistos na Figura 7. Como podemos observar pelos valores das entropias encontrados nas simulações, nem mesmo tentar criar uma lista com as regras mais obtidas nas respectivas posições é suficiente para encontramos soluções ótimas, ou seja, para se conseguir soluções aceitáveis, temos que submeter o autômato à algum procedimento, como no nosso caso, usar o algoritmo do Resfriamento Simulado.

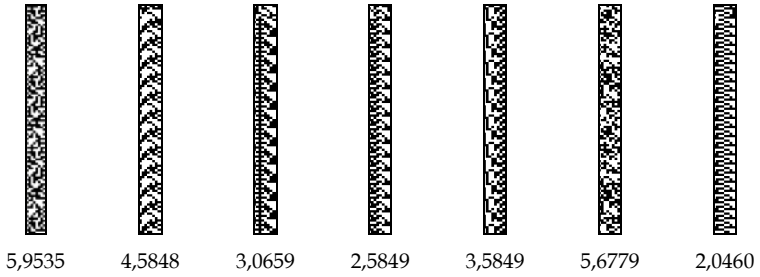


Figura 7. Tentativa de criação de heurística das posições das soluções ótimas.

5.3. Comparação com Algoritmos Genéticos

Tomassini (2001) fez experimentos para gerar a chave de criptografia usando autômatos celulares bastante "entrópicos" gerados a partir de Algoritmos Genéticos, ao invés do Resfriamento Simulado, como proposto na simulação mostrada.

O princípio básico dos Algoritmos Genéticos (AGs) baseia-se em uma analogia direta com a Genética Natural, manipulando conceitos como população de indivíduos, que representam, por sua vez, possíveis soluções do problema a ser resolvido. A cada indivíduo é associado um valor de adaptação (*fitness*), e para aqueles com maior adaptação é dada a oportunidade de reprodução. São selecionados os melhores indivíduos daquela geração. Esses indivíduos são então submetidos a operações como *crossing-over* (um novo indivíduo é criado através da combinação de partes de dois ou mais indivíduos) e mutação (um novo indivíduo é criado por pequenas mudanças arbitrárias sobre os indivíduos). Os AGs podem ser aplicados em diversas áreas. A área de Otimização Combinatória é naturalmente "contemplada" por este algoritmo, já que a tarefa de gerar um indivíduo de melhor adaptação pode ser associada à tarefa de encontrar a melhor solução para um problema.

Tomassini chegou à conclusão de que apenas combinando as regras 90, 105, 150 e 165 em posições sortidas, se obtém uma solução satisfatória. As quatro regras obtidas pelos testes realizados com os AGs são consideradas boas (estão presentes na maioria das soluções ótimas), porém não são suficientes para se chegar a um resultado satisfatório. Devemos combiná-las com outras regras. Como vimos, o resultado usando o *Simulated Annealing* mostrou que a posição das regras também é de fundamental importância, ou seja, não adianta, como concluiu Tomassini, pegarmos as regras consideradas boas e escolhermos, aleatoriamente, suas posições. A Figura 8 mostra alguns resultados sorteando as regras propostas por Tomassini em posições aleatórias.

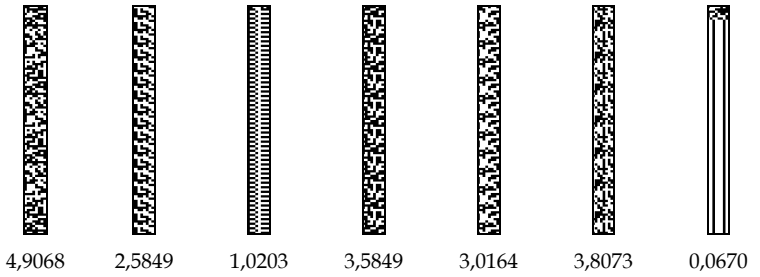


Figura 8. Resultados com a combinação das regras que geram soluções consideradas satisfatórias por Tomassini {90, 105, 150 e 165} em posições aleatórias.

6. Considerações Finais

Nesse trabalho, nós abordamos a utilização de autômatos celulares como sendo chave de cifras de fluxo. Para gerar autômatos que sejam boas soluções como chave, ou seja, a chave seja aleatória suficiente, utilizamos o algoritmo de Resfriamento Simulado.

Realizamos exaustivos testes que renderam tabelas, as quais nos mostram exemplos de soluções, com as regras e suas posições. Com isso, podemos facilmente observar que não é necessário somente acharmos regras "boas" e colocá-las sortidamente em qualquer posição. Para gerarmos boas seqüências aleatórias para serem chaves de uma criptografia de fluxo, devemos descobrir uma combinação de regras e as posições corretas para cada uma delas. Dessa forma, encontrar boas soluções não é uma tarefa simples, como propôs Tomassini na sua solução utilizando Algoritmos Genéticos, que concluiu que apenas sorteando uma regra dentro do conjunto {90, 105, 150, 165} para qualquer posição, se obtém uma solução aceitável.

Vale lembrar que realizamos testes utilizando autômatos celulares unidimensionais de 8 *bits*, que nos permitiu "rodar" 1024 transições de tempo sem que a seqüência começasse a se repetir, gerando uma chave de 8192 *bits*, desprezando a configuração inicial. É uma solução extremamente pequena, mas que já nos prova que obtivemos um resultado correto, e, assim, que, agora, somente necessitaríamos gerar autômatos maiores (16 *bits*, 32 *bits* e casos bidimensionais) que, na prática, poderiam ser utilizados em casos reais.

O que queremos enfatizar, principalmente, com o desenvolvimento do método proposto é a grande dificuldade de encontrar um modo de obter seqüências que sejam aleatoriamente suficientes para serem boas chaves para uma cifra de fluxo.

Referências Bibliográficas

Wolfram, S. (1986) "Cryptography with Cellular Automata", Advances in Cryptology: Crypto'85 Proceedings, Lecture Notes in Computer Science, 218: 429-432.

Nandi, S., Kar, B. K. and Chaudhuri, P. (1994) "Theory and applications of cellular automata in cryptography", IEEE Trans. Computers, 43:1346-1357.

Wolfram, S. (2002) "A New Kind of Science". Wolfram Media, Champaign, IL.

Menezes, A., van Oorschot, P. and Vanstone, S. (1996) "Handbook of Applied Cryptography". CRC Press.

Carvalho, L. A. V. de (1989) "Síntese de Redes Neuronais com Aplicações a Representação do Conhecimento e à Otimização". Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Shannon, C. E. (1948) "A Mathematical Theory of Communication", Bell Syst. Tech. J., 27: 379-423, 623-656.

Tomassini, M. and Perrenou, M. (2001) "Cryptography with cellular automata". Applied Soft Computing, 1:151–160.