UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

CURSO DE ENGENHARIA DE COMPUTAÇÃO

INTELIGÊNCIA ARTIFICIAL

N-Rainhas com Algoritmo Genético e Simulated Annealing

Alunos:

Alexandre Luz
Jaime Dantas
Ramon Fava

Professor:

Allan Robson Silva Venceslau

Natal, 17 de abril de 2016

Sumário

Problema das N-Rainhas

O problema das N-Rainhas, já descrito em Eight Queens Puzzle é um problema combinatório exponencial. Este artigo tem por interesse mensurar o aumento da complexidade com o aumento das dimensões, descrever uma heurística consistente, descrever deduções obtidas em nossas experiências e apresentar 3 algoritmos do tipo Hill Climbing. Hill Climbing é equivalente à técnica do Gradient Descent. As aplicações desses algoritmos são várias, citaremos a seguir algumas:

Para o Gradiente Descendente temos ampla utilização na área de aprendizagem de máquinas. Muitas vezes algumas versões do gradiente descendente são extremamente promissoras como a versão estocástica.

O Hill Climbing com algumas modificações se torna uma boa ferramenta,como é muito simples e fácil de implementar alguns experimentos de otimização utilizam o Hill Climbing para fazer um comparação com o que está sendo desenvolvido[1].

Solução Implementada

A programação principal dos algoritmos foi realizada em C++ com a criacação de duas classes principais: *genetico.h* e *annealing.h.* N arquivo *main.h* foram chamadas cada classe com seus respectivos construtores como os dados passados por atribuição direta de entrada (usado a entrada de argumentos na função *main).* A Sequencia de entrada de variáveis foi padronizada de forma que fosse possível chamar esse programa usando outra linguagem passando somente os parâmetros de entrada e lendo a saída. A saída de nosso programa é um vetor com as posições das rainhas em cada linha e coluna, assim como adotado como padrão de cromossomo. A figura abaixo mostra a estrutura de nosso programa em C++.

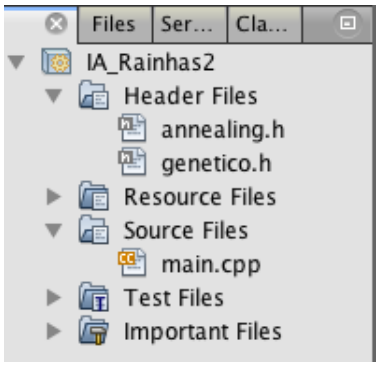

Figura 1 - Estrutura do Programa em C++

Foi criado um programa em PHP para fazer o papel de interface entre o HTML e o C++. No programa HTML foram criados o tabuleiro para mostrar a solução que é passado para ele pelo PHP. A interface final de nosso programa em HTML é apresentada na figura abaixo.

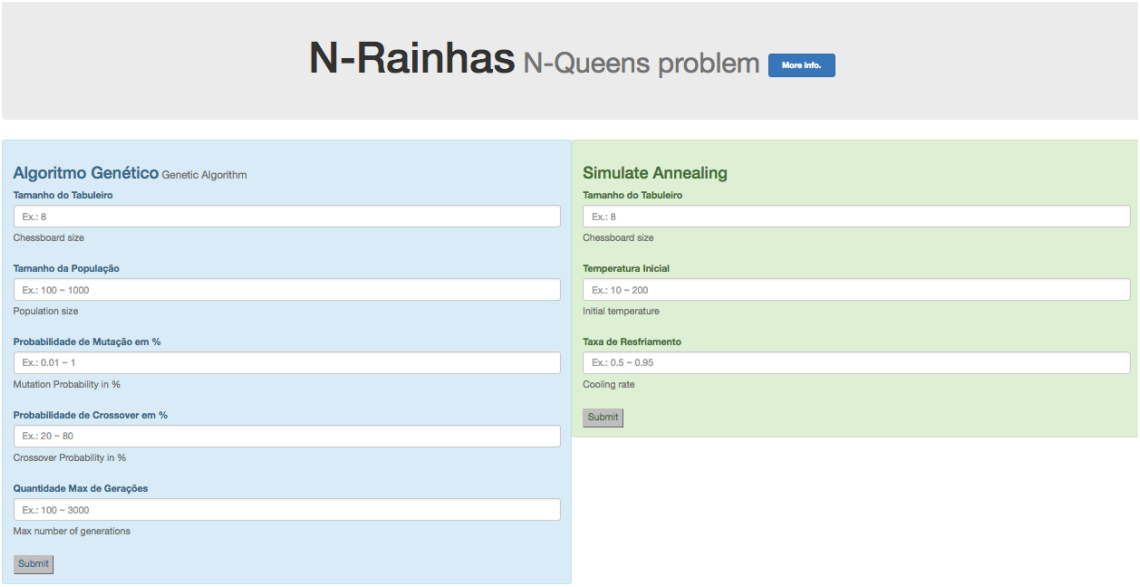

Figura 2 - Interface Gráfica

Na interface gráfica é possível selecionar qual algoritmo queremos executar assim como passar os devidos parâmetros. A figura abaixo mostra a solução de um exemplo de algoritmo genético.

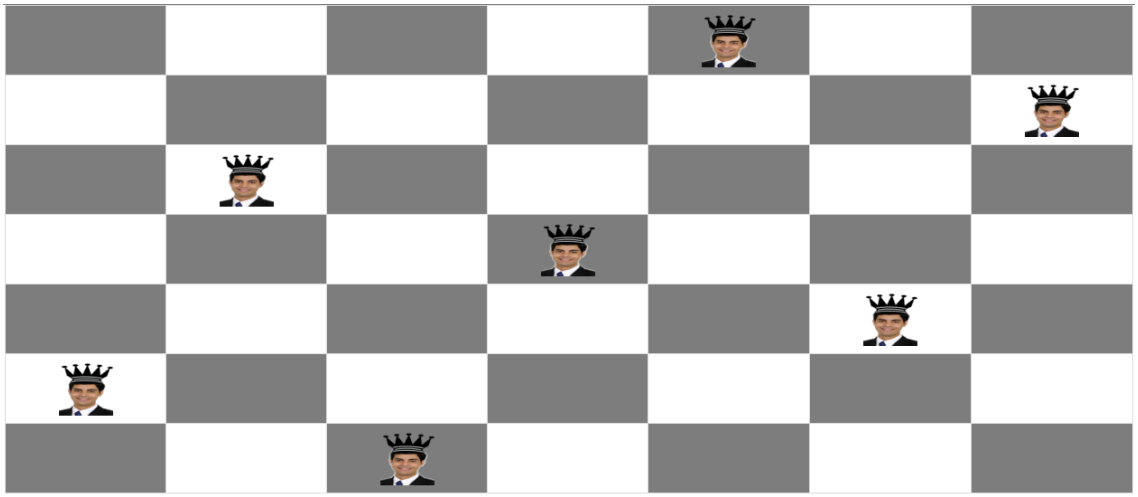

Figura 3 - Solução Algoritmo Genético

Quando ocorre colisões, o respectivo tabuleiro é mostrado na cor vermelho, assim como mostra o exemplo abaixo onde as rainhas linha 0 coluna 3 e 4, linha 1 coluna 2 e 6, linha 4 coluna 5, linha 6 coluna 8 ocorrem colisões.

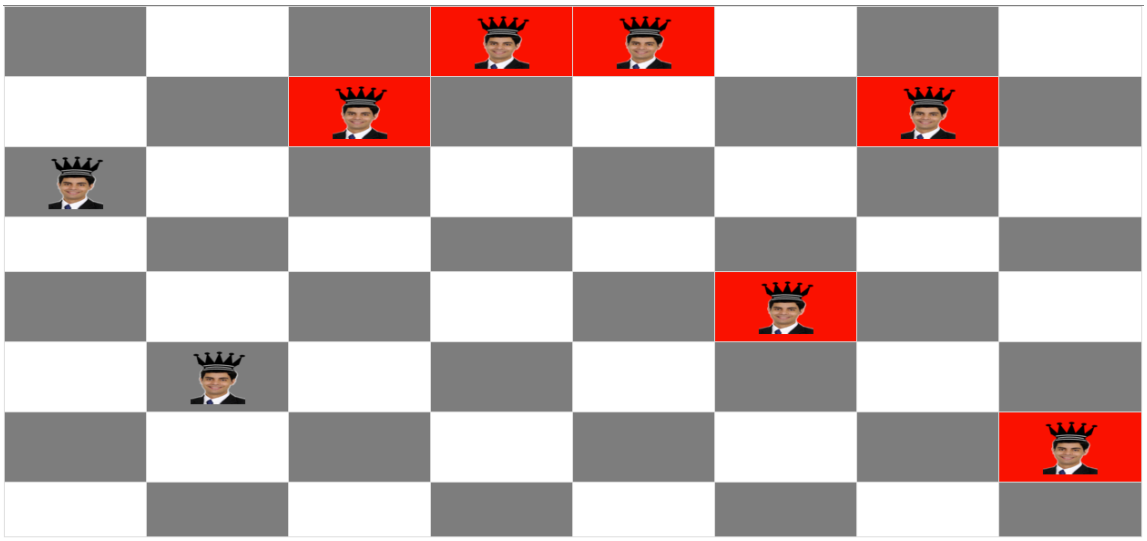

Figura 4 - Exemplo de Colisões com Algoritimo genético

O programa está disponível online para acesso nos endereços abaixo. Ressaltamos que a disponibilidade dos links está sujeita a erros no acesso aos servidores. Em outras palavras, verifique a disponibilidade do link com os administradores do site.

http://alexandreluz.ddns.net/~alexandre/n-rainhas/

http://jaimedantas.ddns.net/index_3.html

Simulated Annealing

É uma técnica de busca local probabilística, em que o resultado é uma solução ótima, porém, não necessariamente a melhor.

O algoritmo funciona substituindo a solução atual por uma solução próxima (na sua vizinhança no espaço de soluções), escolhida de acordo com uma função objetivo e com uma variável T (temperatura). Quanto maior for T, maior a componente aleatória que será incluída na próxima solução escolhida. À medida que o algoritmo progride, o valor de T é decrementado, começando o algoritmo a convergir para uma solução ótima, necessariamente local.

Para resolver o problema das N Rainhas, que consiste em dispor n rainhas em um xadrez n por n, configuramos um vetor inicial em que cada posição contém o lugar onde a rainha está disposta na respectiva coluna. por exemplo, em um xadrez 4x4, com um vetor inicial [0,1,2,3], as rainhas estão dispostas na diagonal principal do tabuleiro.

Começamos a busca a partir de uma solução inicial qualquer. Damos valores para a temperatura inicial, taxa de resfriamento, tamanho n do xadrez. O procedimento principal consiste em um loop que a cada iteração, gera aleatoriamente um único vizinho da solução corrente.

A função objetivo consta em avaliar o número de colisões entre as rainhas dado uma configuração do tabuleiro. ela retornará um valor negativo, pois estamos buscando o valor mínimo.

A cada geração de um novo vizinho, é testada a variação Δ do valor da função objetivo (diferença de energia), isto é, Δ = f (próximo) – f (atual), onde temos as seguintes situações:

- $\Delta > 0$: implica que a nova solução é melhor que a anterior. Assim o vetor resultado recebe os valores da nova solução.

- $\Delta = 0$: Caso de estabilidade, não há redução de energia. A solução é indiferente.
- $\Delta < 0$: A aceitação desse tipo de solução é mais provável a altas temperaturas e bastante improvável a temperaturas reduzidas. Para reproduzir essas características, geralmente usa-se, para calcular a probabilidade de se aceitar a nova solução, uma função conhecida, que é dada por e^(-Δ/temperatura), chamado de temperatura e que regula a probabilidade de soluções com pior custo.
    - Gera-se um número aleatório retirado de uma distribuição uniforme no intervalo [0, 1].
    - Se este número for menor ou igual a "p", aceita-se a solução.
    - Se for maior que "p", rejeita-se a solução.

A temperatura assume inicialmente um valor elevado. Após um número fixo de iterações (o qual representa o número de iterações para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura é gradativamente diminuída por uma razão de resfriamento α, tal que T = Tinicial * α^t, sendo $0 < \alpha < 1$ e t o número de iterações. Como esse procedimento se dá no início, há uma chance maior de se escapar de mínimos locais e, à medida que T se aproxima de zero, o algoritmo se comporta como o método de descida, uma vez que diminui a probabilidade de se aceitar movimentos que possa piorar.

O procedimento é finalizado quando a temperatura chega a um valor próximo de zero ou quando o número de colisões for zero. Encontrando, assim, um resultado otimizado.

Resultados

Foram realizadas 100 execuções consecutivas com os mesmos parâmetros para que poderemos traçar tendências de padrões. Usando a linguagem de programação C++, foi criado um programa para realizar todas os testes de forma automática variando um parâmetro por vez e deixando os outros parâmetros fixados. A saída do programa foi a média de colisões para as 100 execuções consecutivas para cada variação de parâmetro.

O primeiro teste realizado foi o de influencia do tamanho do tabuleiro na solução gerada. Foram fixados para esse teste a temperatura inicial de 100 e o fator de resfriamento em 0.95. Foram testados tabuleiros de tamanhos entre 4 e 100 rainhas. Analisando o gráfico abaixo, podemos concluir que a função que rege a relação entre tamanho de tabuleiro e colisões é linear. É possível notar que para tabuleiros pequenos a quantidades de colisões tende a zero, e para tabuleiros muito grandes, a quantidade de colisões também tende a aumentar linearmente.

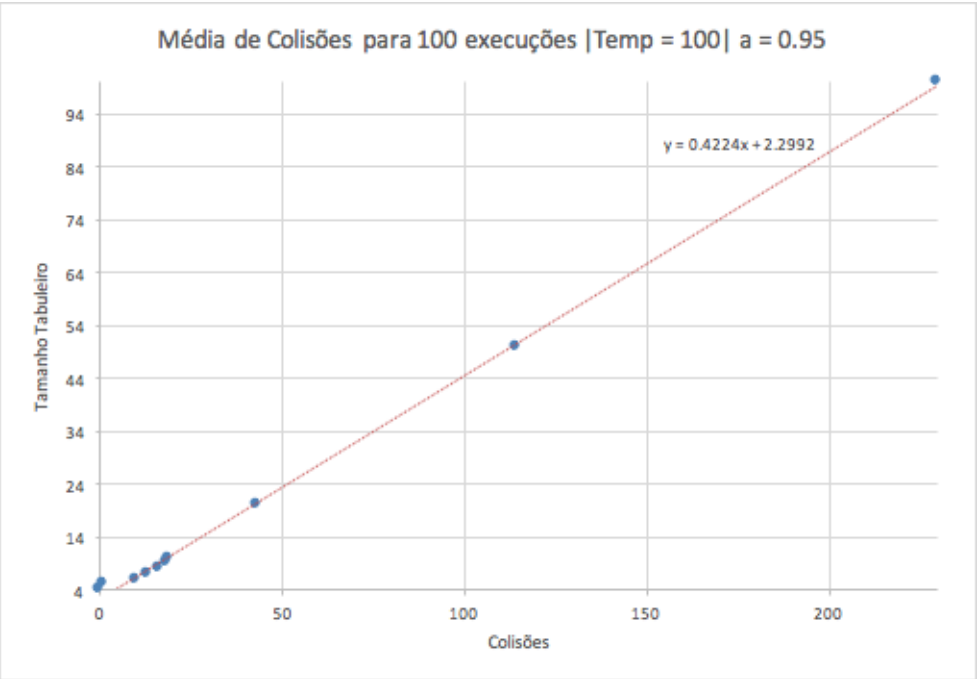

Figura 5 - Variação de Tabuleiro

O fator de resfriamento a tem um papel importantíssimo na qualidade da solução gerada pelo nosso programa. É ela que limita a quantidades de execuções de nosso programa, e geralmente varia de zero à um. Para nossos testes, foram realizados testes com a constante de resfriamento variando de 0.9 até 0.9999. O gráfico abaixo mostra o resultado obtido. Podemos concluir que quando a temperatura de resfriamento tente a 1, a quantidade de colisões tente a zero. Entretendo, para obtermos essa solução ideal, exigiríamos um poder de processamento muito elevado. Quando realizados os testes com $a=0.9999$ o tempo de execução dos testes foi muito alto, com alguns testes durando mais de 3 minutos de execução. Para esse teste foram fixados a temperatura inicial de 1000 e o tabuleiro de 7 rainhas.

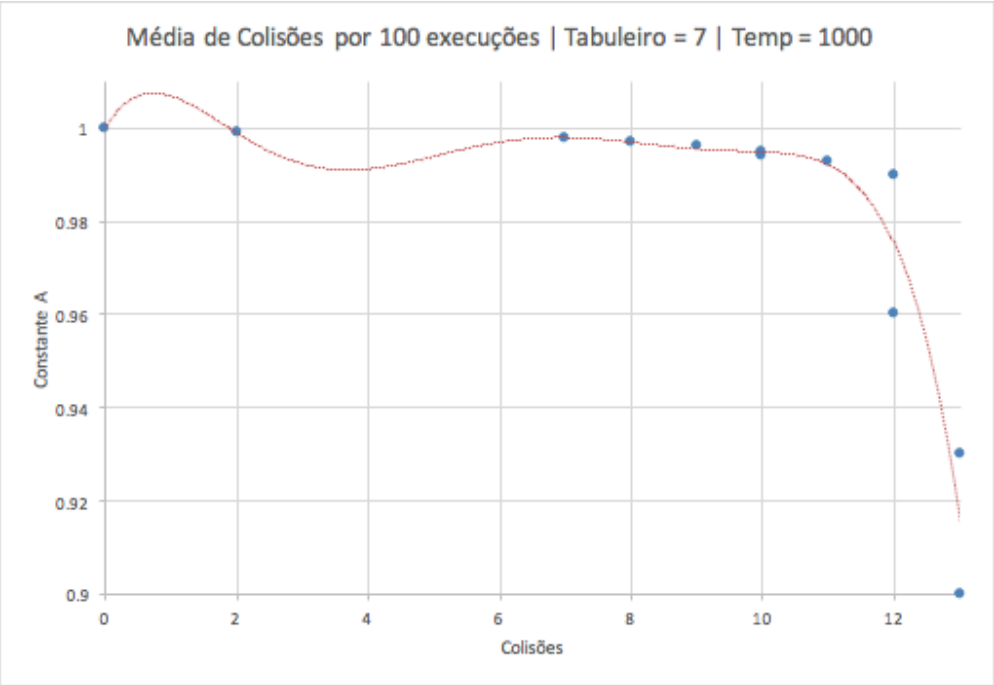

Figura 6 - Variação da taxa de resfriamento

O último teste realizado foi o de temperatura inicial. Foram testadas temperaturas de 1 até 100.000 e os resultados analisados. O resultado obtido como temperatura ideal, ou seja, a que gera menos colisões, foi de $T = 1000$. Para esse teste foram fixados a constante de resfriamento de 0.999 e o tabuleiro de 7 rainhas.

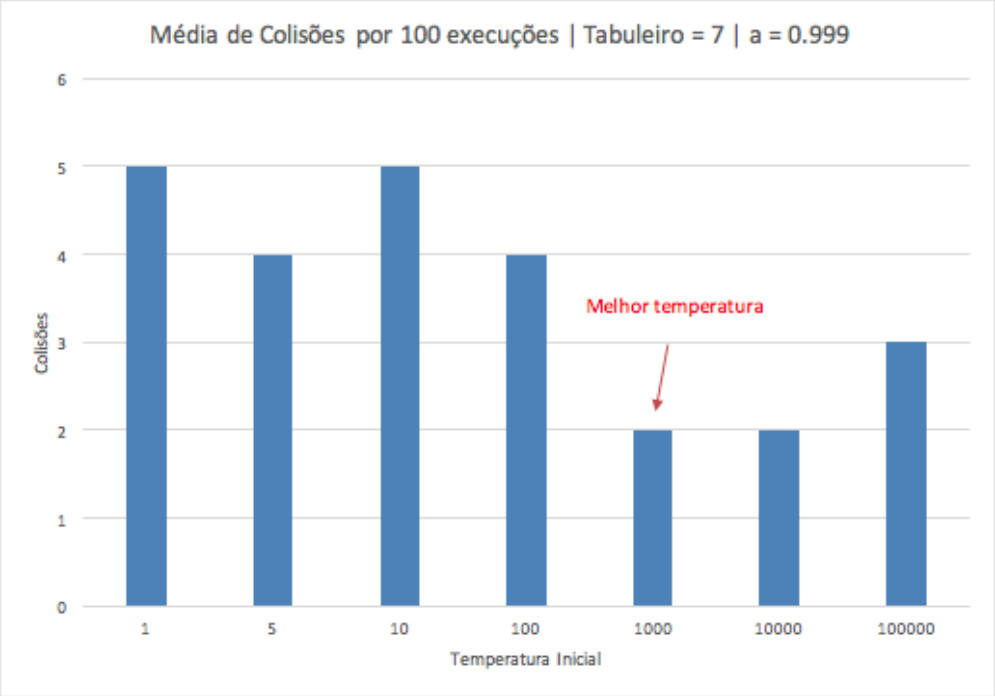

Figura 7 - Variação da Temperatura Inicial

De forma geral, o simulate annealing apresentou uma média de execuções elevada, assim como o excessivo uso de memória e processamento em nossos testes. Em alguns testes, foram usados mais de 600 MB de memória RAM quando executado os testes mais pesados do simulated annealing.

Algoritmo Genético

Algoritmos Genéticos (AG) são implementados como uma simulação de computador em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente e é realizada por meio de gerações. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados ou mutados para formar uma nova população. A nova população então é utilizada como entrada para a próxima iteração do algoritmo.

Para achar o máximo global, essa técnica utiliza:

• Exploração, processo de visitar pontos inteiramente novos no espaço de busca. Necessário para explorar regiões desconhecidas do espaço de busca. Utilizada nas funções crossover e mutação.

• Prospecção – Processo de explorar o espaço de busca utilizando de informações de pontos anteriormente visitados a fim de encontrar melhores pontos. Utilizada na função seleção natural.

Utilizaremos a mesma modelagem das n rainhas do algoritmo anterior para esse.

O programa começa com as escolhas do tamanho do cromossomo, tamanho da população, probabilidade de mutação e probabilidade de crossover.

A probabilidade de mutação é menor do que 1%. Se for elevado vira uma busca aleatória. A mutação previne a permanência em espaços de busca limitado.

A probabilidade de crossover fica entre 60% a 99%.

Assim, cria-se uma população inicial. Depois, avalia-se cada indivíduo dessa população para saber quais os mais “adaptados”, menores números de colisões entre as rainhas.

Os principais métodos da seleção natural são: roleta, torneio, amostragem universal estocástica. Foi utilizado o torneio, onde foi escolhido aleatoriamente 2 elementos, e entre eles, escolhido o mais adaptado.

Na função crossover são escolhidos aleatoriamente 2 indivíduos da população e gerado dois novos indivíduos com a combinação desses dois escolhendo-se um ponto de corte aleatório.

Na função mutação. cada posição do cromossomo é avaliado em torno de uma probabilidade para saber se ocorrerá ou não a mutação, caso ocorra, será mudado aleatoriamente o valor que encontra-se nessa posição.

Ao final volta para o começo do algoritmo avaliando-se os novos indivíduos.

Resultados

Foram-se avaliado o desempenho e os resultados obtidos do algoritmo genético, alterando apenas uma característica do algoritmo e mantendo-se todas as outras constantes, foi possível observar o comportamento do mesmo em várias situações diferentes, e a partir daí pode-se ver como cada elemento do algoritmo influencia nos resultados obtidos.

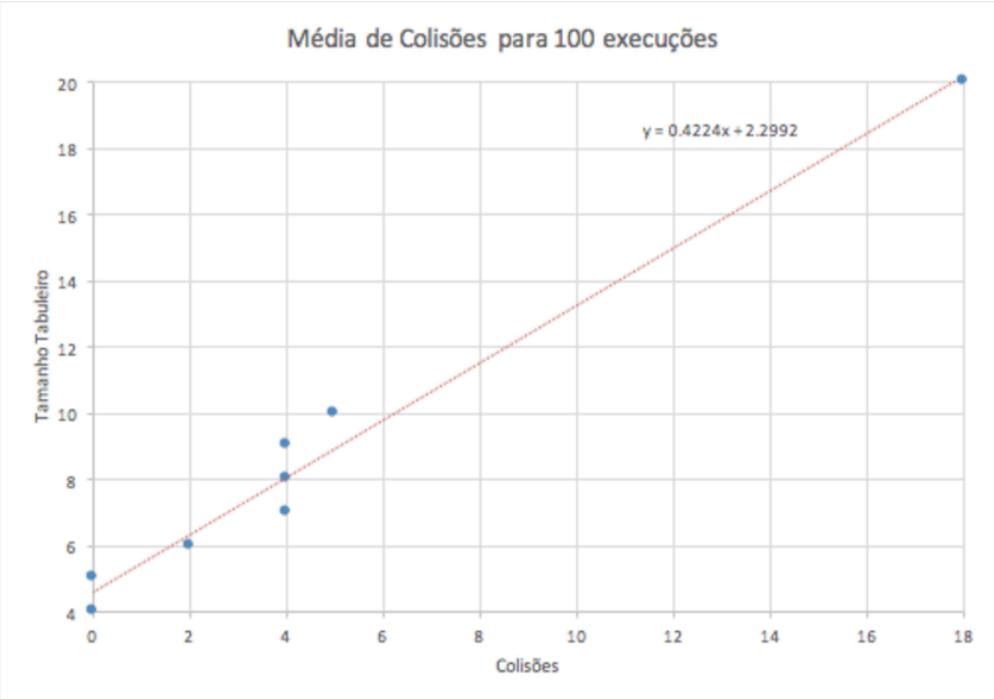

Figura 8 - Variação no Cromossomo

Variando-se apenas o tamanho do tabuleiro, ou seja, o tamanho do cromossomo, pode-se observar que a média de colisões aumenta de forma linear.

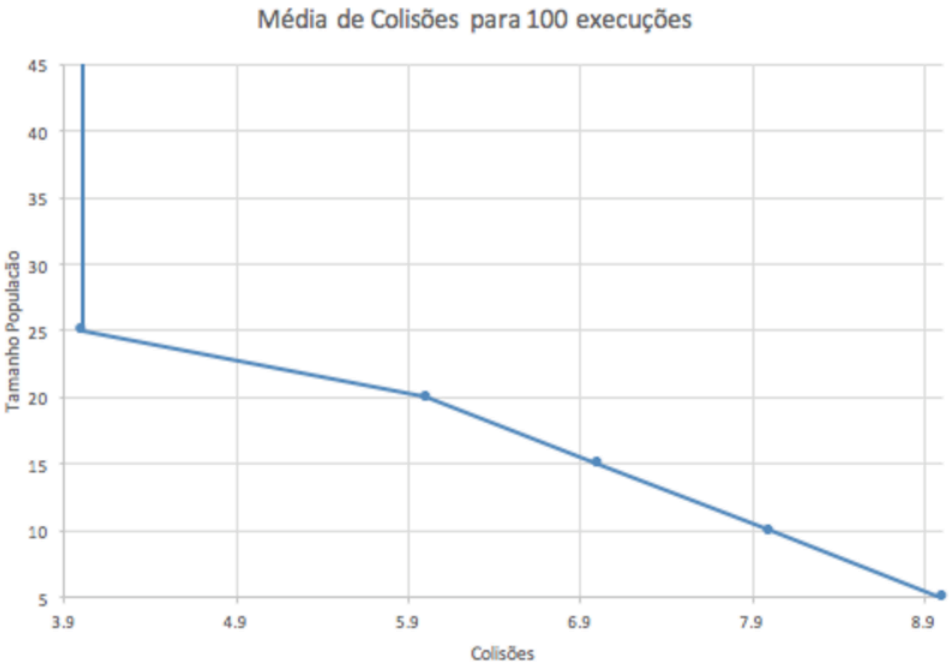

Figura 9 - Variação na População

Variando-se apenas o tamanho da população por outro lado podemos observar uma variação de forma linear para a quantidade de colisões até uma população em torno de 25 cromossomos, e a partir daí a quantidade média de colisões torna-se praticamente constante.

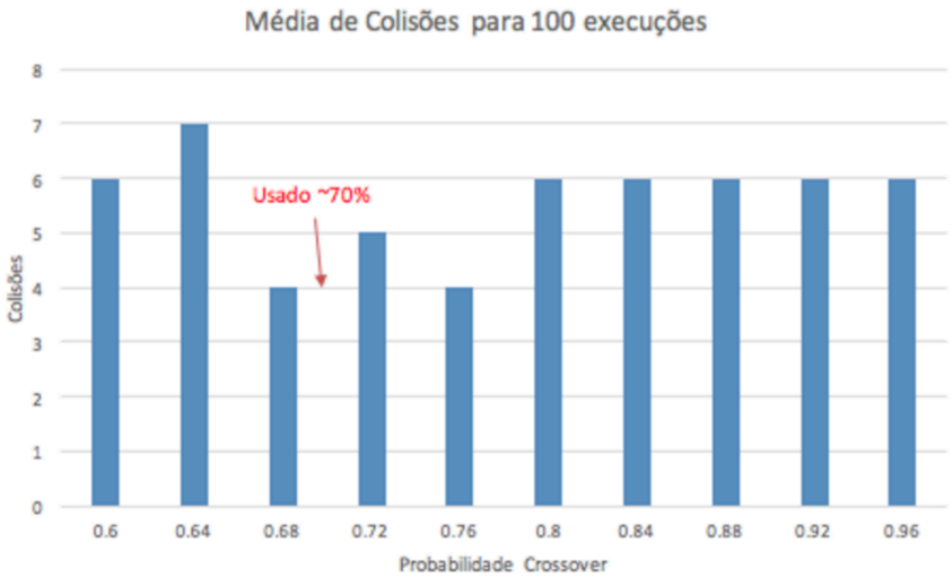

Figura 10 - Variação na Probabilidade de Crossover

A probabilidade de crossover, como já foi dito antes, garante a diversidade de cromossomos na população, fazendo cruzamentos entre cromossomos. Nesse caso, pode-se observar que uma probabilidade em torno de 70% seria o ideal, como o próprio gráfico demonstra.

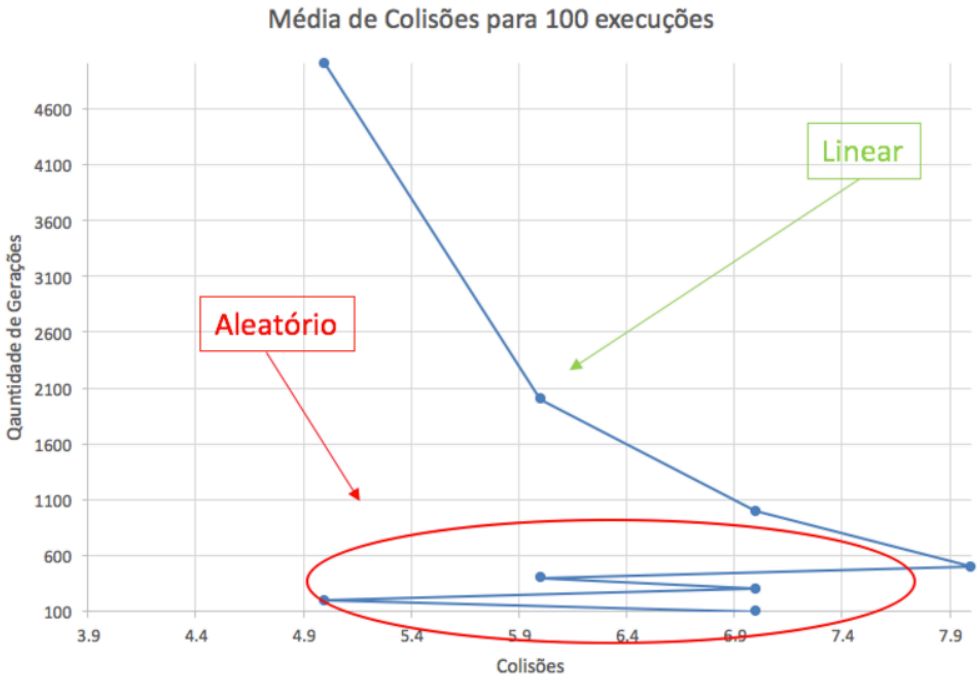

Figura 11 - Variação na Quantidade Máxima de Gerações

Em contraste com os outros elementos, a quantidade máxima de gerações teve pouca ou nenhuma influência na média de colisões para quantidades abaixo de 600, ou seja, a média de colisões ficava inconstante, aleatória. Entretanto a partir de 600 já se é possível observar um comportamento mais linear na quantidade de colisões com o aumento progressivo das gerações.

Por fim, pode-se dizer que para que o algoritmo genético funcione de forma eficaz e o mais rápido possível, mantendo-se uma precisão boa, é preciso levar em consideração vários fatores, desde a complexidade do problema a ser resolvido, até o processamento disponível do computador a ser usado. Uma mistura de tamanhos variados dos elementos do algoritmo é o recomendado para alcançar tal resultado, procurando encontrar um equilíbrio entre os valores, para que nenhum fique demasiadamente maior que os outros.

Referências Bibliográficas

[1] Wikpedia, **Problema das N-Rainhas.** Disponível em https://pt.wikipedia.org/wiki/Problema_das_N-Rainhas Acesso em 17 de abril de 2016.