Relatório Inteligência Artificial – Problema das N-Rainhas utilizando o Algoritmo Simulated Annealing e Problema do Caixeiro Viajante utilizando Algoritmos Genéticos

Daniel Rosendo de Souza

[1]Instituto Federal do Ceará – IFCE Campus Maracanaú

daniel.rosendos@hotmail.com

Resumo. *Este relatório tem como objetivo explicar e mostrar de forma pratica a implementação do problema das N-Rainhas utilizando como solucionador o algoritmo do Recozimento Simulado (Simulated Annealing) e a implementação do algoritmo genéticos para resolução do problema do caixeiro viajante. Vamos mostrar como foram feitos as implementações e os resultados de cada algoritmo.*

1. Introdução

O problema das n-rainhas é o problema matemático de dispor n rainhas em um tabuleiro de xadrez de dimensões NxN, de forma que nenhumas delas seja atacada por outro, para tanto é necessário que duas damas quaisquer não estejam numa mesma linha, coluna ou diagonal (para qualquer $n \geq 4$).

O problema do caixeiro viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-Difícil inspirado na necessidade dos vendedores em realizar entregar em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

Os algoritmos genéticos (AGs) são um dos vários métodos que se utilizam para a resolução de problemas complexos. Este método tem por base um processo iterativo sobre uma determinada população fixa, denominados por indivíduos, que representam as várias soluções do problema. Esta técnica advém do processo de evolução dos seres vivos demonstrada por Darwin.

Da mesma forma que os sistemas biológicos, ao longo da sua evolução, tiveram que se moldar às alterações ambientais para a sua sobrevivência, os AGs acumulam a informação sobre o ambiente com o intuito de se adaptarem ao novo meio. Tal informação funciona como um sistema de triagem para a obtenção de novas soluções exequíveis.

O método dos algoritmos genéticos é muito utilizado devido à simplicidade de operação, eficácia pela determinação de um máximo global e aplicabilidade em problemas onde se desconhece o modelo matemático ou onde o mesmo se torna impreciso em funções lineares e não-lineares.

O recozimento simulado, ou Simulated Annealing, é uma meta-heurística para otimização que consiste numa técnica de busca local probabilística, e se fundamenta numa analogia com a termodinâmica.

A metaheurística usada é uma metáfora de um processo térmico, dito annealing

ou recozimento, utilizado em metalúrgica para obtenção de estados de baixa energia num sólido. O processo consiste de duas etapas: na primeira, a temperatura do sólido é aumentada para um valor próximo de 1100 °C, que é a temperatura de início de transformação da fase perlítica em austenita; na segunda, o resfriamento deve ser realizado lentamente até que o material se solidifique, sendo acompanhado e controlado esse arrefecimento. Nesta segunda fase, executada lentamente, os átomos que compõe o material organizam-se numa estrutura uniforme com energia mínima. Isto resulta em que os átomos desse material ganhem energia para se movimentar livremente e, ao arrefecer de forma controlada, dar-lhes uma melhor hipótese de se organizarem numa configuração com menor energia interna, para ter uma redução dos defeitos do material, como resultado prático.

De forma análoga, o algoritmo de Arrefecimento Simulado substitui a solução atual por uma solução próxima, escolhida de acordo com uma função objetivo e com uma variável T (dita Temperatura, por analogia). 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. Uma das principais vantagens deste algoritmo é permitida testar soluções mais distantes da solução atual e dar mais independência do ponto inicial da pesquisa.

2. Implementação

As implementações dos algoritmos listados acima, foram feitas utilizando a linguagem python na sua versão 2.7 para a implementação do problema das N-rainhas, utilizando o recozimento simulado como solucionador e a linguagem python na sua versão 3.7, para a implementação do problema do Caixeiro viajante, utilizando algoritmos genéticos para solução, e escolha de deus versões diferentes para solucionar os problemas, foram devido a bibliotecas especificas das versões. Para codificar foi utilizando a IDE PyCharm para implementar os códigos.

A implementação do Recozimento Simulado foi baseada em 3 formas, uma classe para os Nós (Que representa os estados ou a disposição das rainhas no tabuleiro, a classe simulated_annealing que é a implementação em si do algoritmo e a classe de teste, para inicializar o projeto. Nosso algoritmo recebe como entrada de dados o Estado Inicial do tabuleiro, a quantidade de iterações que o algoritmo irá executar para achar a solução, a quantidade de perturbações que a solução irá receber, a quantidade de sucessos por iteração e o fator de redução da temperatura. Ao ser executado, no console é exibido o tabuleiro no formato inicial, a temperatura inicial, a quantidade de iterações até chegar no resultado final, a temperatura final, o estado final, a quantidade de ataques das rainhas e por fim é imprimido o tabuleiro no estado final. Toda execução o estado inicial é aleatório, as rainhas são dispostas de forma aleatório.

A implementação do Algoritmo Genético para resolver o problema do caixeiro viajante, foi criado em apenas um arquivo, nele é implementado a classe Cidades , que serve para criar os caminhos das cidades, de forma aleatória, para verificar a funcionalidade do algoritmo, a classe de Fitness que basicamente é o treinamento para verificar quem é a melhor população de caminhos do problema. E adiante é a implementação de funções para rodar o algoritmo, como criar as rotas entre as cidades, inicializar a população, ranquear as rotas para saber as melhores rotas, a classe de seleção para selecionar a população de elitização, o cruzamento das população, a nova população com os novos genes, mutação do genes, para criar a mutação genética, mutação da

população para aplicar a nova mutação genética na população, função de pegar a nova geração calcular ela, e a inicialização do AGs.

3. Resultados Obtidos

Para o problema das N-Rainhas utilizando o recozimento simulado, inicialmente iremos fazer um teste para um numero de 100 iterações, com no máximo 100 perturbações, com 10 de sucessos e com um alpha de 0.99, para um tabuleiro 4x4 abaixo temos um resultado:

```
Estado inicial
[[1 0 0 0]
 [0 0 0 0]
 [0 1 0 0]
 [0 0 1 1]]
Temperatura inicial: 80
Iteracao: 101| Temperatura: 29.282587301858328
Estado: [0, 2, 1, 2] | Iteracao: 101 | Ataques: 3
Numero de Sucessos: 10
Estado Final
[[1 0 0 0]
 [0 0 1 0]
 [0 1 0 1]
 [0 0 0 0]]
```

Com 100 iterações o algoritmo não foi capaz de encontrar uma solução, totalizando em um total de 3 ataques de rainhas.

Agora para um total de 500 iterações utilizando os mesmos parâmetros para perturbação sucessos e mesmo tamanho de tabuleiro, obtemos o resultado abaixo:

```
Estado inicial
[[0 0 0 0]
 [0 0 0 0]
 [0 1 0 1]
 [1 0 1 0]]
Temperatura inicial: 58
Iteracao: 501| Temperatura: 0.3810880164600474
Estado: [0, 2, 3, 1] | Iteracao: 501 | Ataques: 1
Numero de Sucessos: 10
Estado Final
[[1 0 0 0]
 [0 0 0 1]
 [0 1 0 0]
 [0 0 1 0]]
```

Por um pouco nosso algoritmo não achou uma solução, totalizando 1 ataque entre as rainhas.

Utilizando-se de mais um teste, porém fazendo 1000 iterações, utilizando os mesmos parâmetros acima, obtemos o resultado abaixo:

```
Estado inicial
[[1 0 0 0]
 [0 1 0 0]
 [0 0 0 0]
 [0 0 1 1]]
Temperatura inicial: 84
Iteracao: 568| Temperatura: 0.28147162659889785
Estado: [2, 0, 3, 1] | Iteracao: 568 | Ataques: 0
Numero de Sucessos: 0
Estado Final
[[0 1 0 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 0 1 0]]
```

Utilizando os mesmos parâmetros acima e aumentando o tamanho do tabuleiro para 8x8 temos o seguintes resultados:

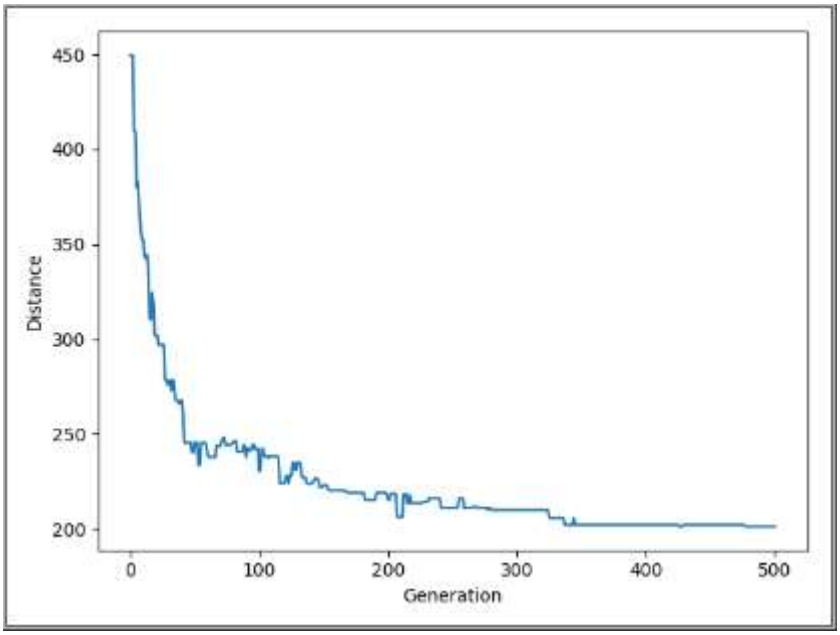
```
Estado inicial
[[0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 1]
 [0 1 0 1 0 1 0 0]]
Temperatura inicial: 71
Iteracao: 514| Temperatura: 0.40936732706936907
Estado: [7, 1, 4, 2, 0, 6, 3, 5] | Iteracao: 514 | Ataques: 0
Numero de Sucessos: 0
Estado Final
[[0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0]]
```

Também com 1000 iterações e um tabuleiro 8x8 o algoritmo foi capaz de encontrar uma solução para o problema das n-rainhas.

Para o problema do caixeiro viajante inicializamos um vetor com a lista de cidades, e criamos uma lista de cidade aleatória, assim chamamos nossa função AGs passando como parâmetro nossa lista de cidades, o tamanho da população que definimos como 100, o tamanho da elite e que definimos como 20, a taxa de mutação de 0.01 e um total de 500 gerações e obtivemos os seguintes resultados mostrados abaixo:

```
Initial Distance: 449.53347038716277
Final Distance: 201.2086290650232
[(24,40), (20,35), (10,41), (0,49), (6,42),
```

Temos como resultado a distancia inicial, a distancia final quando passou pelo algoritmo, e a lista com os novos caminhos.

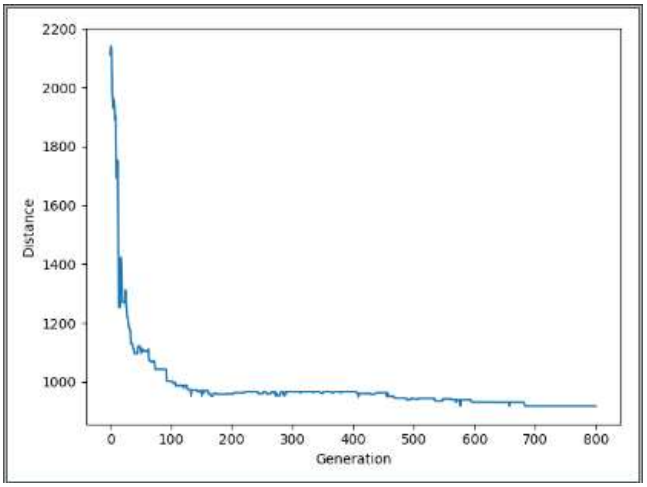


E o gráfico de como se comportou as novas gerações, a partir de que geração convergiu para o melhor caminho (trajeto) pelo caixeiro viajante. Teste obtidos a partir de um caminho de 0 a 50 de distância.

Para outro teste de distancia de 0 a 200 utilizando os mesmos parâmetros, e aumentando o tamanho de gerações para 800 obtemos o seguinte resultado:

```
Initial Distance: 2113.1957997554805
Final Distance: 917.2895728740674
[(83,110), (86,185), (13,185), (0,159), (20,80),
```

E graficamente obtemos:



É possível notar que na geração próximo de 600 , foi dado alguns picos de convergência até chegar próximo a geração 700 onde o resultado convergiu.