Instituto Federal Catarinense
Campus Rio do Sul
Ciência da Computação

Daniel Rodrigo Marconatto

BUSCA DE SOLUÇÃO PARA PUZZLE-8 UTILIZANDO ALGORITMO GENÉTICO

Setembro
2019

1. Introdução

O sistema terá como objetivo encontrar ao menos uma solução (não necessariamente a melhor) para o problema de quebra-cabeça deslizante de 8 peças, também conhecido como Puzzle-8.

Este quebra-cabeça consiste em uma matriz 3 x 3 onde cada posição é composta por uma peça (neste trabalho cada peça será tratada como um número de 0 a 8), apenas uma posição ficará vazia, esta será usada para movimentação das outras peças que serão deslizadas para esta posição "vazia". Apenas as peças adjacentes à posição "vazia" poderão ser deslizadas para ela, considerando as bordas do tabuleiro.

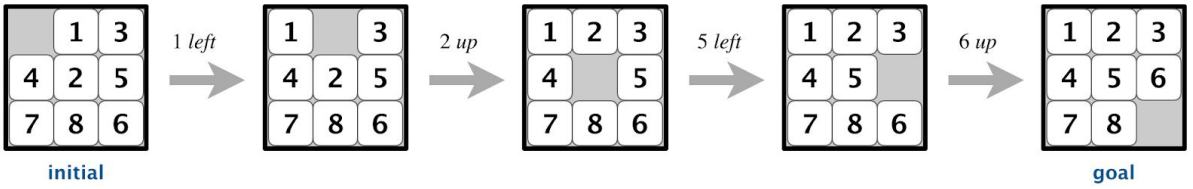

Figura 1: Movimentação das peças no Puzzle-8.

O objetivo do quebra-cabeça é deixar as peças do tabuleiro em ordem crescente, movendo as peças adjacentes, uma de cada vez, à posição "vazia", como é possível ver na Figura 1.

2. Representação cromossomial

Levando em consideração que o objetivo deste trabalho não seja o de achar a melhor solução para o problema, mas sim alguma solução em geral, a representação cromossomial adotada foi a seguinte:

$$\text{Cromossomo} = \{X\}$$

Onde "X" representa a direção à qual a posição "vazia" será movida, utilizando valores de 0 à 3 para determinar a direção, onde 0 → Cima, 1 → Baixo, 2 → Esquerda e 3 → Direita.

Inicialmente foi pensada uma representação cromossomial composta por 9 fatores, que representaria a situação atual do tabuleiro em forma de vetor, porém essa ideia foi deixada de lado pois as operações de cruzamento poderiam acabar gerando problemas nas movimentações das peças, como ignorar as bordas do tabuleiro ou até gerar estados em que a movimentação seria impossível (fazendo movimentações na diagonal por exemplo).

3. Função de avaliação

A função de avaliação do indivíduo foi baseada na função desenvolvida por Florêncio Junior e Guimarães (Figura 2), porém, foi utilizado o número de inversões diretas no lugar de "NumInversoes", o motivo será explicado mais à frente.

$$f(n) = 36 * NumPecasTrocadas + 18 * DistaciaManhattan + 2 * NumInversoes$$

Figura 2: Função de avaliação de florêncio Junior e Guimarães.

Esta fórmula é baseada em três heurísticas, sendo elas:

- ***NumPecasTrocadas:*** representa o número de peças fora do lugar;
- ***DistanciaManhattan:*** representa a soma do custo de movimentação para colocar cada peça em seu devido lugar;
- ***NumInversoes:*** a inversão acontece quando uma peça tem como sucessor alguma peça com valor inferior ao dela. Aqui foi utilizado apenas o número de inversões diretas, ou seja, apenas quando o valor da peça que está sendo analisada menos o valor da peça sucessora seja igual a 1. A heurística *NumInversoes* calcula a troca de posições com toda e qualquer peça com valor inferior à atual sucessora à ela, já a inversão direta calcula apenas com a peça sucessora à qual está sendo analisada.

4. Operadores genéticos

Devido à representação cromossomial conter apenas um elemento, é perceptível que o cruzamento ou "cross-over" será impossível de realizar, portanto o principal operador genético utilizado foi a mutação já que o nosso cromossomo possui apenas um gene, que representa a direção do movimento da peça.

Porém para sanar o problema de falta de hereditariedade foi utilizada a técnica de clonagem, que funciona basicamente em selecionar os melhores indivíduos, cloná-los e substituir os indivíduos medianos pelo clones. A escolha de substituição dos indivíduos medianos e não dos "piores" indivíduos se deu pelo fato de os "piores" indivíduos desta geração poderem gerar novas gerações de indivíduos "melhores" que os desta geração.

5. Testes

Por questão de padronização, utilizamos o mesmo estado inicial para a execução do algoritmo genético e possibilitar assim, calcular o número de gerações necessárias para encontrar uma solução (lembrando que o objetivo é encontrar alguma solução, e não a melhor solução possível). É possível ver o estado inicial na Tabela 1:

2	6	0
5	7	3
8	1	4

Tabela 1: Estado inicial do Puzzle-8.

Os testes foram realizados analisando a média do número de gerações obtida através de 10 repetições utilizando populações de 10, 50 e 100 indivíduos, e cada população tendo 1, 2, 3 ou nenhum par(es) de clones, tratado como "parentes", como pode ser visto no Gráfico 1.

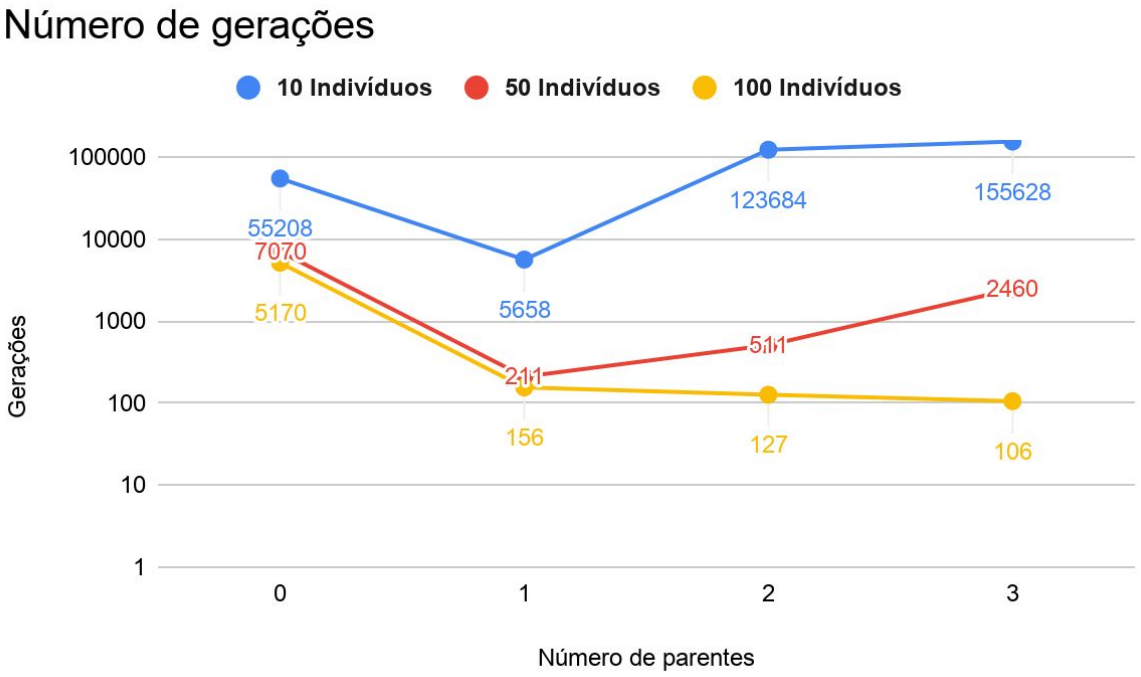

Gráfico 1: Número de gerações.

Ao observar o Gráfico 1, é possível notar que sem a utilização de parentes, o número de gerações necessárias para alcançar algum resultado foi bem próximo nas populações de 50 e 100 indivíduos, porém com a população de 10 indivíduos houve um salto no número de gerações, isso se deu pelo fato de que uma população menor necessita de mais gerações para alcançar resultados diferentes e possivelmente melhores.

Analisando os resultados das populações de 50 e 100 indivíduos com 1, 2 e 3 pares de clones, é possível ver que a utilização de clones diminui consideravelmente o número de gerações necessárias, porém não é bom aumentar o número de clones sem aumentar proporcionalmente a população, senão pode haver "falta" de mutação, fazendo com que a população não evolua com a chegada de novas gerações.

6. Código

Aqui será apresentado apenas alguns métodos do código do algoritmo genético para busca de solução do problema Puzzle-8, mas o código completo está disponível no repositório do GitHub[1].

a. Objeto

Inicialmente foi criado um objeto chamado No que contém os atributos necessários, sendo eles: a matriz (3x3) e o fitness, ambos do tipo inteiro. No mesmo objeto foi criado os métodos Getter e Setter para cada atributo.

```
1  public class No {
2      ArrayList<Integer> passos;
3      int[][] matriz;
4      int fitness;
5  }
```

Tabela 2: Objeto No;

b. Geração de população inicial

```
1  public static No[] geraPopulacaoInicial(No nodo) {
2          Random random = new Random();
3          No[] population = new No[populacaoSize];
4          ArrayList<Integer> moves = movimentos(nodo.getMatriz());
5          for (int i = 0; i < population.length; i++) {
6              No individuo = new No();
7              int move = moves.get(random.nextInt(moves.size()));
8              ArrayList<Integer> lista = nodo.getPassos();
9              lista.add(move);
10             individuo.setMatriz(movimenta(nodo.getMatriz(), move));
11             individuo.setPassos(lista);
12             population[i] = individuo;
13         }
14         return population;
15 }
```

Tabela 3: Método para criação da população inicial.

Aqui basicamente é criado um array do objeto No e populado com as novas matrizes resultantes dos movimentos disponíveis da atual.

[1] Link do repositório do GitHub: https://github.com/DanielMarconatto/Genetic-Algorithm-Puzzle-8

c. Função de avaliação

Aqui calculamos o número de peças fora do lugar, a distância de Manhattan e número de inversões diretas e retornamos aplicando a fórmula.

```
1  public static int Fitness(No nodo) {
2      int numPecasTrocadas = calculaPecas(nodo.getMatriz());
3      int distanciaManhattan = manhattan(nodo.getMatriz());
4      int numInversoes = inversoes(nodo.getMatriz());
5      return ((36 * numPecasTrocadas) + (18 * distanciaManhattan) + (2 * numInversoes));
6  }
```

Tabela 4: Aplicação da função de avaliação.

d. Seleção de população

Neste método, recebemos a população, ordenamos os indivíduos por melhor fitness, então buscamos os indivíduos medianos (pela ordenação estão mais ao meio do vetor), e substituímo-os por clones dos melhores indivíduos, e então reordenamos a população.

```
1  public static No[] selecionaPopulacao(No[] populacao) {
2      populacao = ordenaPopulacao(populacao);
3      for (int i = 0; i < parentes; i++) {
4          populacao[(populacao.length / 2) + i] = populacao[i];
5      }
6      populacao = ordenaPopulacao(populacao);
7      return populacao;
8  }
```

Tabela 5: Método de seleção da população.

e. Criação de nova população

Agora, geramos a nova população baseada na atual geração, onde cada indivíduo vai escolher uma direção para mover a peça vazia, baseado nos movimentos disponível da matriz do mesmo indivíduo da geração passada.

```
1  public static No[] geraPopulacao(No[] populacao) {
2      Random random = new Random();
3      No[] population = new No[populacaoSize];
4      for (int i = 0; i < population.length; i++) {
5          ArrayList<Integer> moves = movimentos(populacao[i].getMatriz());
6          No individuo = new No();
7          int move = moves.get(random.nextInt(moves.size()));
8          individuo.setMatriz(movimenta(populacao[i].getMatriz(), move));
9          ArrayList<Integer> lista = populacao[i].getPassos();
10         lista.add(move);
11         individuo.setPassos(lista);
12         population[i] = individuo;
```

```
13          }
14          return population;
15      }
```

Tabela 6: Geração da nova população.

f. Ordenação de população

Para realizar a ordenação da população, foi utilizado o método Insertion Sort considerando o fitness de cada indivíduo.

```
1   public static No[] ordenaPopulacao(No[] populacao) {
2           for (int i = 1; i < populacao.length; i++) {
3               No aux = populacao[i];
4               int j = i;
5               while ((j > 0) && (populacao[j - 1].getFitness() > aux.getFitness())) {
6                   populacao[j] = populacao[j - 1];
7                   j -= 1;
8               }
9               populacao[j] = aux;
10          }
11          return populacao;
12  }
```

Tabela 7: Método para ordenação da população.

7. Conclusão

Analisando principalmente o resultado dos testes realizados, foi possível concluir que a utilização de algoritmos genéticos para resolução de problemas de tabuleiro como o quebra-cabeça de oito peças traz grandes vantagens ao sistema, como agilidade e rapidez em problemas com “infinitas” soluções.

Porém é necessário muita atenção, cautela e estudo na determinação da representação cromossômica e da função de avaliação, já que estes influenciam diretamente nas chances de sucesso da aplicação.

Deixo a sugestão para objeto de estudo a adição de novos valores heurísticos à função de avaliação utilizada como base para este trabalho, deixando-a mais complexa e exata, garantindo um caminho mais “assertivo”, diminuindo o número de gerações necessárias para encontrar uma solução. Outro ponto a ser estudado é a representação cromossômica, talvez utilizando outras formas de representação, seja possível trabalhar melhor com a questão de “cross-over” entre indivíduos, não necessitando assim da técnica de clonagem, e podendo gerar uma otimização das mutações.

8. Referências

Junior, Nelson F.; Guimarães, Frederico G.; Problema 8-Puzzle: Análise da solução usando Backtracking e Algoritmos Genéticos .Disponível em:<http://www.decom.ufop.br/menotti/paa111/files/PCC104-111-ars-11.1-NelsonFlorencioJunior.pdf>. Acesso em: 22 Setembro 2019.