Algoritmos genéticos para resolução de quebra cabeças

Para estudo dos algoritmos genéticos escolhemos o problema de **resolução de quebra cabeças**, pouco explorado pela literatura (encontramos apenas a referência [1]).

O problema, em sua forma mais simples, consiste em pegar imagens de mesmo tamanho que compunham uma imagem maior, e restaurar a imagem completa com os pedaços de imagens nas posições corretas. Para simplificar o estudo do problema, não implementamos nenhum tipo de técnica de processamento de imagens (a **dissimilaridade** como utilizada em [1], parece funcionar bem para medir o quanto duas imagens se encaixam), ao invés disso, para implementar a função fitness consideramos **o número de vizinhos errados** que cada peça tem, portanto, cada peça pode contribuir com inteiros de 0 até 4 para a fitness total, que é a soma das fitness de todas as peças do quebra cabeça. Vale ressaltar que é fácil utilizar a dissimilaridade em nossa implementação, visto que ambos têm a mesma natureza: **zero é a combinação perfeita** e quanto mais uma peça estiver diferente dos vizinhos maior seria esse valor, permaneceria, portanto, um **problema de minimização**.

Além disso, o problema é substancialmente sensível em termos de complexidade do número de combinações válidas possíveis: por exemplo um quebra-cabeça 3 x 3 possuí 9! combinações (362880), enquanto um problema 5 x 5 possui 25! ou 1.551121e+25 combinações válidas

Algoritmo genético utilizado

Abaixo temos uma descrição abstrata do funcionamento do algoritmo genético utilizado:

1. Cria população inicial com tamanho especificado de forma aleatória, calcula fitness e ordena.
2. Seleciona indivíduos da população para fazer crossover de acordo com as técnicas e parâmetros de selection.
3. Itera de 2 a 2 nos pais selecionados fazendo crossover dos mesmos (o número de filhos que o crossover gera pode variar com a técnica) e estes são integrados a lista de próxima geração.
4. Caso a lista de próxima geração for menor que a população atual completamos com os clones dos melhores indivíduos.
5. Aplicamos mutação na lista da próxima geração
6. Calculamos a fitness da lista de próxima geração e a ordenamos.
7. Aplicamos o replacement da população (mantendo o tamanho da população, dando prioridade aos melhores).
8. Caso o melhor indivíduo não satisfaça o critério de parada voltamos a iterar em 2.

Os parâmetros disponíveis para variação deste algoritmo são:

- Tamanho da população
- Técnica de seleção: roleta ou torneio
- Técnica de crossover: split randômico, pedaços alternados, melhores pedaços por fitness ou combinação das melhores linhas
- Técnica de mutação: swap nos pedaços, swap linhas e colunas
- Técnicas de replace: elitism, steady state ou extermination;
- Número de indivíduos na seleção
- Tamanho do torneio
- Crossover rate
- Mutation rate
- Mutation swap: porcentagem de swaps em um indivíduo;
- Método de swap: forma mais aleatória ou considerando as fitness
- Steady state rate

Principais Resultados

Os resultados obtidos não foram tão bom como esperados pelo grupo - esperávamos conseguir obter resultados mais razoáveis para instâncias maiores. Na maioria das configurações do algoritmo genético foi razoavelmente fácil resolver instâncias pequenas como 4x4, já 7x7 e o 8x8 dificilmente convergiram rapidamente - já que a maioria do processo do algoritmo genético se deu de forma aleatória, e não conseguimos chegar à um crossover que resolvesse o problema de forma eficiente, aproveitando as melhores partes dos pais.

O algoritmo genético melhorou de forma notória (convergência com menos iterações) quando tiramos um pouco da "aleatoriedade" da mutação de swap fazendo com o que o "swap" tivesse tendência de acontecer com mais frequência nas peças "mais erradas", ou com o maior número de vizinhos errados. Comparação que pode ser vista na **Figura 1**.

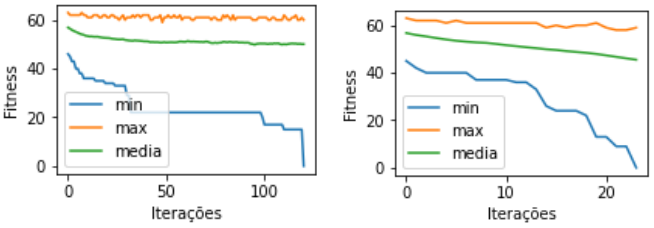

Figura 1 - Comparação entre as duas formas de swap implementadas, a da esquerda é totalmente aleatória, enquanto a da direita leva em consideração as peças com vizinhos mais errados para a operação de "swap"

O problema é que mesmo com a mutação melhorada ainda não conseguimos bons resultados para instâncias maiores, pois ela tem a "falha" de considerar a otimização local das peças e não de forma global para se chegar na melhor solução.

Na **Figura 2** temos o gráfico da execução de uma instância de tamanho 8x8

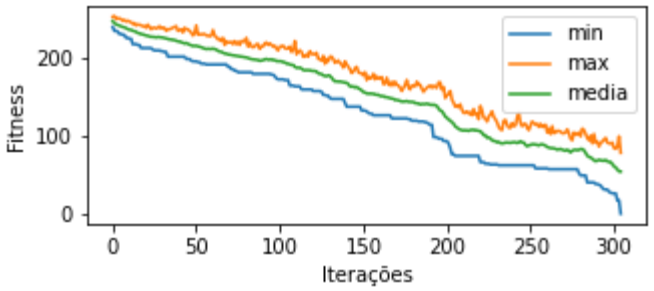

Figura 2 - Execução do algoritmo em um dos experimentos para uma instância 8x8, levando 15s para execução total até a convergência, utilizando a mutação melhorada.

Neste gráfico da Figura 2, é nítido como o algoritmo atua de forma lenta nas primeiras iterações, sendo quase uma reta decrescente e no final há uma grande queda do melhor indivíduo devido ao problema estar quase resolvido, bastando apenas que a mutação faça as trocas corretas para encontrar a melhor solução.

Conclusões

Concluímos que apesar do problema possuir uma premissa simples, há muita dificuldade em implementar um algoritmo genético que o resolva de forma viável e eficiente para maiores instâncias. A falta da implementação de um crossover melhor para uma convergência mais rápida, deixou o trabalho incomparável com a referência [1], o que porém não inviabilizou o nosso estudo de algoritmos genéticos.

Bibliografia

[1] Sholomon, Dror & David, Eli & Netanyahu, Nathan. (2014). Genetic Algorithm-Based Solver for Very Large Multiple Jigsaw Puzzles of Unknown Dimensions and Piece Orientation.

[2] RUSSEL, S.; NORVIG, P. Artificial Intelligence: a modern approach. Prentice Hall. 3rd edition, 2010.