Alunos: Hojin Ryu, Igor Dourado e Igor Vicente **T1 - Inteligencia Artificial**

1. Configuração

1.1 CODIFICAÇÃO DO PROBLEMA

Na entrada do comando, recebemos o nome do arquivo de entrada, o tamanho da população inicial e o percentual de mutação (de 0 a 100). Quebramos a cada linha do txt e colocamos num array de ranking.

1.2 GERAÇÃO DA POPULAÇÃO INICIAL

Criamos um array de um tamanho inserido na entrada e inserimos cromossomos que foram feitos aleatoriamente com random.sample em python.

1.3 FUNÇÃO DA APTIDÃO

Nossa função de aptidão calcula um ***alpha*** que é 1/quantidadeDePares, e esse alpha é multiplicado pela posição daquele aluno no ranking dele (de forma invertida), isso é feito para os dois, somado e dividido por 2. Por exemplo:

3
1 2 1 3
2 3 2 1
3 1 3 2
1 3 1 2
2 1 2 3
3 2 3 1

Para o exemplo acima o *alpha* é 1/3. Para o aluno 1 da manhã a aptidão ficaria em 100% pois a prioridade dele é correspondida pelo aluno 2 que também o tem como prioridade, o cálculo fica assim:

AlphaManhã = 3 [*posição do '2' no ranking invertida*] * 1/3 [*alpha*] = 3/3 = 1 (100%)
AlphaTarde = 3 [*posição do '1' no ranking invertida*] * 1/3 [*alpha*] = 3/3 = 1 (100%)
Aptidão = (*AlphaManhã* + *AlphaTarde*) / 2 [*manhã e noite*] = (1+1) / 2 = 1 (100%)

Esse foi um exemplo para o primeiro do ranking do aluno 1 da manhã, esse processo deve se repetir para todos os pares descritos em um cromossomo, dessa forma ao fim teremos o somatório das aptidões desse cromossomo, logo basta dividir esse somatório de aptidões do cromossomo pelo número de pares. Dessa forma teremos uma aptidão de no máximo 1 (100%) e no mínimo *alpha*, quando todos estão em par com suas piores prioridades.

1.4 OPERADOR DE SELEÇÃO

A operação utiliza o método de elitismo, ou seja, mantem o melhor cromossomo da população atual na próxima geração.

1.5 OPERADOR DE CRUZAMENTO

Utilizamos crossover PBX (Position Based Crossover). Com o array da população, para cada 2 cromossomos, criamos 2 filhos. Em seguida, selecionamos posições aleatórias e, para cada uma delas, inserimos o valor na posição do cromossomo_1 na mesma posição no filho_2 e o valor na posição do cromossomo_2 na mesma posição no filho_1. Para o restante, copiamos os demais elementos para as posições vagas na ordem que aparecem nos seus cromossomos pais.

1.6 OPERADOR DE MUTAÇÃO

Utilizamos o percentual de mutação fornecido na execução do programa para definir quantas mutações serão realizadas na população. Ou seja, em uma população de tamanho 10 e percentual 50, poderá ocorrer no mínimo 0 e no máximo 5 mutações, essa quantidade é gerada randomicamente entre 0 e (percentual da mutação * tamanho da população).

Tendo a quantidade de mutações que ocorrerão na população a cada mutação é selecionado um cromossomo da população para sofrer mutação, podendo ser selecionado em outra seleção inclusive.

A mutação ocorre trocando 10% das posições do cromossomo de lugar com outras dele mesmo, para isso utilizamos a função *embaralhar(cromossomo, 0.1*qtdPares).*

1.7 CRITÉRIO DE PARADA

Temos 3 critérios para parar o algoritmo:
- Quando encontrou solução ótima (aptidão 100%);
- Quando a convergência dos valores é maior que 95%;
- Quando houver 1000 iterações sem melhora na aptidão do melhor cromossomo;

2. Testes e Resultados

Utilizando os arquivos de teste fornecidos no Moodle testamos com tamanhos de população diferentes e diferentes coeficientes de mutação. Ademais, a cada execução registramos a população inicial e as demais geradas pela reprodução ao longo do processo de execução do algoritmo no escrita.txt para acompanhar a evolução do algoritmo.

Para o relatório iremos adicionar apenas 4 testes feitos com o exemplo (pares10.txt), mas a reprodução dos testes podem ser facilmente replicadas e testadas alterando os parâmetros que usamos para a execução no terminal com outros arquivos de testes, tamanho de população e coeficiente de mutação, como demonstrado a seguir:

$ python app.py <***arquivoInput***>*.txt* <**tamanhoPopulacao**> <**PercentualMutacao**>
output: geração **[% de aptidão]** (pares formatados) **array:** [pares em array]

Exemplo para o arquivo pares10.txt com tamanho da população sendo 100 e percentual de mutação 0:
Execução: $ python app.py *pares10.txt* 100 0
Critério de parada: **Convergência dos valores maior que 95%**
Output:
24 **[95.0%]** (1, 10) (2, 9) (3, 8) (4, 7) (5, 6) (6, 5) (7, 4) (8, 3) (9, 2) (10, 1) **Array:** [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Exemplo para o arquivo pares10.txt com tamanho da população sendo 100 e percentual de mutação 25:
Execução: $ python app.py *pares10.txt* 100 25
Critério de parada: **Muitas iterações sem melhora**
Output:
138 **[95.0%]** (1, 10) (2, 9) (3, 8) (4, 7) (5, 6) (6, 5) (7, 4) (8, 3) (9, 2) (10, 1) **Array:** [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Exemplo para o arquivo pares10.txt com tamanho da população sendo 20 e percentual de mutação 0:
Execução: $ python app.py *pares10.txt* 20 0
Critério de parada: **Convergência dos valores maior que 95%**
Output:
21 **[90.0%]** (1, 3) (2, 9) (3, 8) (4, 7) (5, 6) (6, 5) (7, 4) (8, 10) (9, 2) (10, 1) **Array:** [3, 9, 8, 7, 6, 5, 4, 10, 2, 1]

Exemplo para o arquivo pares10.txt com tamanho da população sendo 20 e percentual de mutação 25:
Execução: $ python app.py *pares10.txt* 20 25
Critério de parada: **Muitas iterações sem melhora**
Output:
120 **[95.0%]** (1, 10) (2, 9) (3, 8) (4, 7) (5, 6) (6, 5) (7, 4) (8, 3) (9, 2) (10, 1) **Array:** [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]