Relatório Inteligência Artificial

Projeto Stocking Problem

Tiago Marques Amaral
2181390

Miguel Ângelo Coito Fialho
2181430

1. INTRODUÇÃO

A minimização de recursos é um aspeto fulcral de qualquer indústria. No caso da indústria têxtil, uma disposição ineficiente implica o desperdício de matéria-prima, por resultar em restos que devido à sua forma e/ou tamanho não podem ser aproveitados.

Posto isto, no âmbito da unidade curricular de Inteligência Artificial foi solicitado o desenvolvimento uma aplicação com vista à mitigação deste problema. O objetivo da aplicação em questão é otimizar a disposição de um conjunto de peças numa superfície, de modo a minimizar os recursos necessários ao seu fabrico.

2. IMPLEMENTAÇÃO

Para resolver o problema descrito acima, a aplicação utiliza os seguintes algoritmos: algoritmo aleatório e algoritmo genético.

2.1 Algoritmo Aleatório

O algoritmo *random* limita-se a explorar o espaço de resultados aleatoriamente, sem ter em conta qualquer informação que possa existir devido a pesquisas passadas.

Logo, a sua implementação consistiu em avaliar um individuo aleatório, que posteriormente vai ser comparado com outros indivíduos aleatórios, durante um certo número de iterações. Ficando no fim desse ciclo de iterações com o melhor individuo de entre os avaliados.

2.2 Algoritmo Genético

2.2.1 Representação do problema

Antes de efetuar a construção do genoma é imperativo que se percorra o genótipo, e nesse sentido também é importante referir que no problema em causa, o genótipo não define onde é que vão ser colocadas as peças, uma vez que, este se trata de uma permutação de números inteiros, que determina a ordem de colocação das peças. No problema em questão o genoma comporta as peças, sendo que cada peça corresponde a um item, e o item é identificado através de um id (que é um número inteiro que corresponde a um valor ASCII, estando assim cada id associado a uma letra) e representado através de uma matriz.

```
Genoma: [14, 2, 11, 4, 8, 7, 5, 10, 3, 1, 13, 6, 0, 9, 12]

Peças: [ O C L E I H F K D B N G A J M ]
```

Fig.1 Genoma de um individuo e o conjunto de peças que está a representar

2.2.2 Representação da solução

Os indivíduos são um aspeto crucial da implementação do algoritmo genético, visto que é a partir destes que se obtém as soluções para o problema. Cada individuo é representado por um conjunto de características sendo uma delas o seu genoma.

O genoma é criado a partir dos elementos aleatórios do genótipo, em que o mesmo não pode ter valores repetidos. Na nossa implementação usamos uma lista que vai conter todos os elementos do genótipo, ao percorrer o genoma adicionamos aleatoriamente o valor do genótipo ao genoma e removemo-lo da lista.

Estes indivíduos vão ser adicionados à população, que por sua vez representa um conjunto de soluções possíveis.

Depois de determinada a solução para o problema, esta é representada através de uma matriz que contem as peças em questão, podendo estas ser identificadas através da letra à qual estão associadas.

```
Matriz:
O O O O O O O O H H F F F F
O O O O O O O O H H F F F F
C C C L L L D D H H N N N N
C C C L L L D D H H G G G
E E K L L L B B H H A J J
E E I I I I B B H H A M M M
```

Fig.2 Matriz com as peças da solução

2.2.3 Função de avaliação

A melhor solução para o problema é encontrada através da avaliação do melhor individuo presente na população.

A função que realiza esta avaliação é a *computeFitness*, que está localiza na classe *StockingProblemIndividual*. Este método começa por percorrer o genoma, para saber que peça é que vai ser adicionada à matriz. Seguidamente, percorremos as colunas da matriz da solução (que foi dimensionada recorrendo à altura e ao comprimento do material), são percorridas primeiro as colunas, porque queremos colocar a peça o mais acima e à esquerda possível, feito isto percorremos as linhas da matriz da solução. Antes de colocar a peça na matriz da solução temos de averiguar em que parte da matriz encaixa a peça, feito isso a peça é colocada.

De realçar que, apesar de todos estes processos serem definidos no *computeFitness* ainda não foi referido como é que os valores que compõem o *fitness* são calculados. O *fitness* utilizado neste algoritmo genético é a soma entre: o número de cortes necessários para modelar o tecido utilizado, e o número de colunas que estamos a usar para colocar o material, sendo que estes dois componentes vão ter uma preponderância diferente aquando da sua soma, para definir o valor do *fitness*.

O método auxiliar que coloca uma peça na matriz, também é responsável por definir o valor da variável 'tamMaxPec', que diz respeito ao número de colunas que estamos a usar para colocar o material, este verifica qual é o valor que está na coluna onde está a ser colocada a peça, se o valor for maior que o do 'tamMaxPec', substitui o valor do 'tamMaxPec' pelo da coluna. Enquanto, o valor da variável 'nCuts' diz respeito ao número de cortes necessários, e é calculado através de dois varrimentos da matriz da solução, para verificar se ocorreram cortes na vertical e na horizontal, no momento em que a matriz está a ser percorrida é comparada a posição atual da matriz com a sua posição seguinte, se as posições forem diferentes significa que houve um corte. Sempre que tal ocorre, o valor da variável 'nCuts' é incrementado. Após a definição dos valores do 'tamMaxPec' e 'nCuts', dá-se a soma dos mesmos, e finalmente obtemos o valor da função de avaliação.

```
NºCortes: 46.0

TamanhoMaxPeça: 8.0

Fitness: 19.4
```

Fig.3 Valores do fitness, do 'tamMaxPec' e do 'nCuts' de uma solução

2.2.4 Operadores genéticos

Na tentativa de obter melhores resultados no desenvolvimento da população, foram utilizados operadores genéticos, nomeadamente operadores de mutação e operadores de recombinação.

2.2.4.1 Mutação

Os operadores de mutação alteram o genoma do individuo, permitindo que outras áreas do espaço de procura sejam exploradas.

2.2.4.1.1 Mutação Swap

São selecionados aleatoriamente dois elementos do genoma, e trocadas as posições entre eles.

Preservando muita da informação adjacente.

Exemplo:

Pai: 9 1 6 2 3 7 4 5 8

Filho: 9 1 5 2 3 7 4 6 8

2.2.4.1.2 Mutação Inversion

São selecionados aleatoriamente dois pontos no genoma, e é invertida a ordem dos genes entre esses dois pontos, de seguida este conjunto de genes é deslocado para qualquer lugar ao longo do tamanho do genoma.

Pai: 0 1 2 3 4 5 6 7

Filho: 0 1 2 3 6 5 4 7

Exemplo: Analisando o cromossoma -> 0 1 2 3 4 5 6 7
Selecionamos dois pontos aleatórios (variáveis cut1 e cut2, tendo em conta que cut2 > cut1), onde consideramos que foram selecionados genes entre o 5º e o 7º elemento do cromossoma (representados a vermelho), respetivamente o cut1 a apontar para a posição 5 e o cut2 a apontar para a posição 7: 0 1 2 3 4 5 6 7

Depois é invertida a ordem desses genes: 0 1 2 3 6 5 4 7

E por fim, este conjunto de genes é deslocado para um lugar ao longo do tamanho do cromossoma. Supondo que pretendemos, deslocar do 5º elemento para o 2º elemento (deslocamento de 3 posições para a esquerda): 0 6 5 4 1 2 3 7

2.2.4.2 Recombinação

Os operadores de recombinação geram dois novos indivíduos (filhos) a partir de outros dois (pais), produzindo assim dois novos genomas.

2.2.4.2.1 Recombinação Order

Começa por copiar uma sequência de genes (selecionada devido às escolhas aleatórias de um ponto de corte inicial e final) do pai2 para o filho1.

Sendo que, os restantes genes são copiados para o filho1: começando no ponto de corte final da sequência copiada; usando a ordem dos genes do pai1; e caso seja necessário, o

genoma do pai1 é percorrido do princípio ao fim, para encontra os genes que ainda não foram copiados.

Exemplo:

Pai 2: 8 4 7 3 6 2 5 1 9 0
Pai 1: 0 ~~1 2 3~~ 4 ~~5 6~~ 7 8 9

Filho 1: 0 4 7 3 6 2 5 1 8 9

A criação do filho2 é similar à do filho1, com a diferença de os papeis do pai1 e do pai2 estarem invertidos.

2.2.4.2.2 Recombinação Cycle

A recombinação cycle identifica um número de ciclos entre dois pais de cromossomas. Para criar o filho1, o primeiro ciclo é copiado do pai1, o segundo ciclo do pai2, o terceiro ciclo do pai1 e assim sucessivamente.

Para criar um ciclo, temos que começar com o primeiro alelo do pai1 e na mesma posição do alelo do pai1 vamos verificar o valor do alelo do pai2, e este valor vai indicar a próxima posição do pai1. Adicionamos este valor ao ciclo e repetimos os passos até chegar novamente ao valor inicial do pai1.

Exemplo:

Pai 1: 0 1 2 3 4 5 6 7 8 9

Pai 2: 0 4 5 9 7 3 8 1 2 6

Filho 1: 0 4 2 3 7 5 6 1 8 9

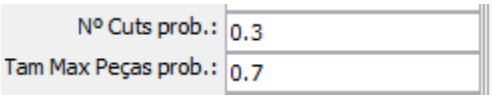
Filho 2: 0 1 5 9 4 3 8 7 2 6

2.3 Extras

2.3.1 Peso dos parâmetros do fitness

O valor do fitness corresponde à soma entre: o número de cortes necessários para alcançar a solução, e o número de colunas que estamos a usar para colocar o material (tam max peças).

Posto isto, é possível ao utilizador definir através da aplicação, qual destes dois elementos vai ter maior peso aquando da sua soma para calcular o fitness.



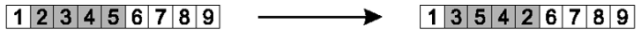
2.3.2 Elitismo

O objetivo deste elitismo é adicionar o melhor individuo da população original à população auxiliar. Esta adição sucede-se depois da ação do método de seleção sobre a população auxiliar, e antes da ação dos operadores genéticos sobre a população auxiliar.

2.3.3 Mutação Scramble

São selecionados dois pontos de corte de forma aleatória, garantindo que o ponto de corte inicial é menor que o ponto de corte final. E a ordem dos genes, presentes no segmento entre os dois pontos, é trocada aleatoriamente.

Exemplo:



2.3.4 Método seleção RankBased

O método de seleção *rank based* atribui um *rank*/posição aos indivíduos em que a primeira posição irá ser o pior individuo e a posição N irá ser o melhor individuo, sendo que N será o total de indivíduos.

Este método de seleção minimiza as convergências num máximo local, faz com que a população não fique estabilizada numa solução que não é ótima.

Como a seleção é feita por base no *rank*/posição e não nos valores do fitness permite evitar que pequenos grupos

de melhores indivíduos influenciem gerações futuras.

A nossa implementação do método de seleção *rank based* permite atribuir *rank* aos indivíduos do pior para o melhor sem comparar com o valor do fitness. Tivemos alguma dificuldade na implementação da probabilidade, não nos foi possível efetuar uma distribuição discreta ou continua para realizar o calculo do mesmo.

3. REALIZAÇÃO DE TESTES

Através da realização de diversos testes onde variamos os parâmetros que compõem a nossa aplicação, esperamos atingir a melhor solução para cada conjunto de dados.

Os resultados obtidos podem variar consoante o computador em que os testes foram realizados, logo é importante referir que os testes foram realizados em dois computadores diferentes.

3.1 Especificações dos computadores utilizados

PC que executou experiências relativas a: mutação

Sistema Operativo	Windows 10 Pro
Processador	Intel(R) Core (TM) i5-4460 CPU @ 3.20GHz 3.20 GHz
Memória RAM	8.00 GB

PC que executou experiências relativas a: população/geração, torneio, recombinação e testes gerais

Sistema Operativo	Windows 10 Pro
Processador	Intel i7-6700k CPU@ 4.0Ghz – 4.2Ghz
Memória RAM	16.00 GB

3.2 Experiências

Antes de analisar os resultados obtidos, importa referir que estas experiências foram realizadas com o elitismo referido nos extras implementado.

3.2.1 Melhor combinação de parâmetros

De modo a realizar os testes da forma mais otimizada possível e obtendo os resultados mais significativos, despois de diversos testes com diferentes combinações de parâmetros e diferentes *dataSets* (nomeadamente os maiores), foram definidos os parâmetros para o ficheiro de configuração:

Runs	30
Population_size	200
Max_generations	50
Selection	*tournament*
Tournament_size	5
Recombination	*order*
Recombination_probability	0.6
Mutation	*swap*
Mutation_probability	0.3

Como é obvio estes parâmetros variam consoante o tipo de experiência em questão, com o objetivo de estudar qual o impacto que têm na amostra.

3.2.2 População/Geração

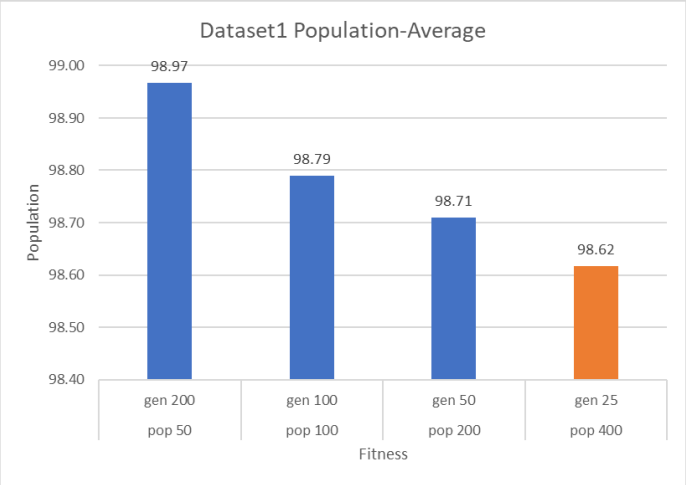
Com estas experiências pretendemos analisar o efeito da variação do tamanho da população e do número de gerações no algoritmo genético.

Para fabricar os testes da população/geração, foram inseridos no ficheiro de configurações os seguintes parâmetros:

```
Population_size: 50, 100, 200, 400

Max_generations: 25, 50, 100, 200
```

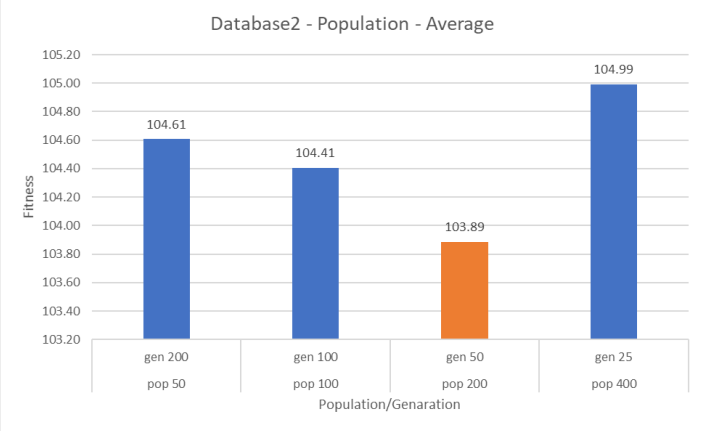
Como existe uma relação da proporcionalidade entre população/geração forma gerados testes para as combinações: 50/200, 100/100, 200/50 e 400/25.

3.2.2.1 DataSet1



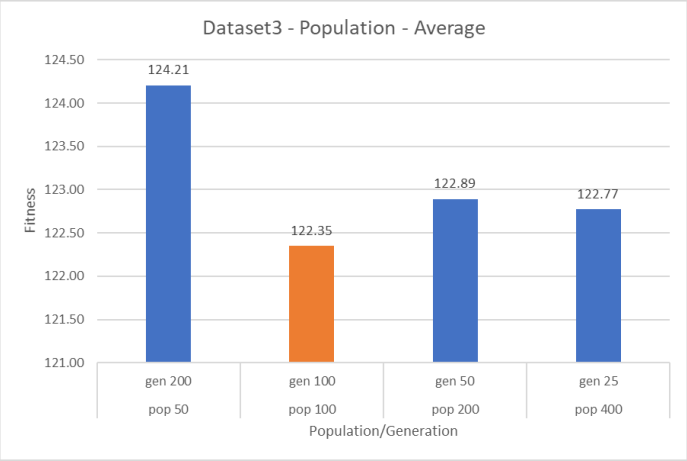
Analisando o gráfico verificamos que os melhores resultados provêm da relação população/geração de 400/25.

3.2.2.2 DataSet2



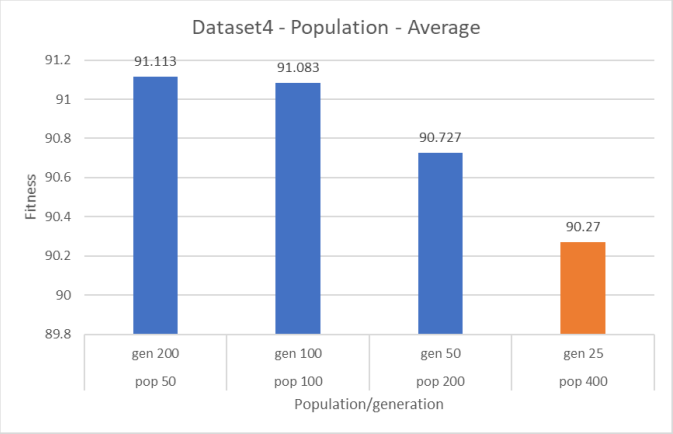
Analisando o gráfico verificamos que os melhores resultados provêm da relação população/geração de 200/50.

3.2.2.3 *DataSet3*



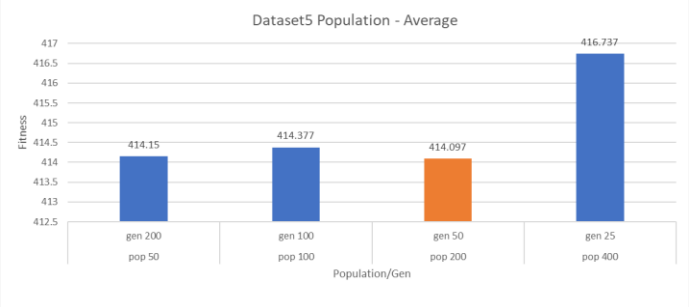
Analisando o gráfico verificamos que os melhores resultados provêm da relação população/geração de 100/100.

3.2.2.4 *DataSet4*



Analisando o gráfico verificamos que os melhores resultados provêm da relação população/geração de 400/25.

3.2.2.5 *DataSet5*



Analisando o gráfico verificamos que os melhores resultados provêm da relação população/geração de 200/50.

Após a realização das experiências neste conjunto de parâmetros, inferimos que não existe uma relação população/geração que apresente universalmente melhores resultados, variando consoante o *dataSet*.

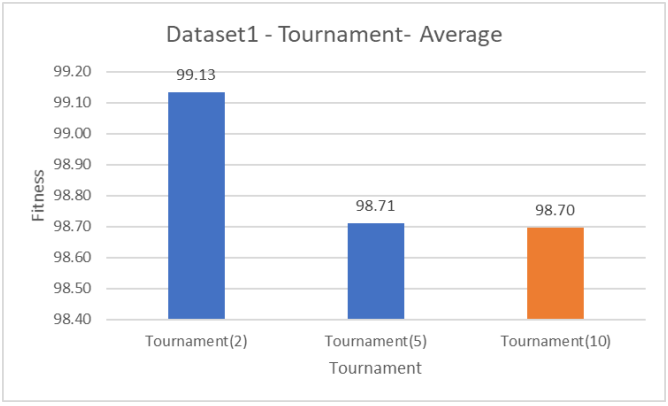
3.2.3 Tamanho Torneio

Com estas experiências pretendemos analisar o efeito da variação do tamanho do torneio no algoritmo genético.

Para fabricar os testes do tamanho do torneio, foram inseridos no ficheiro de configurações os seguintes parâmetros:

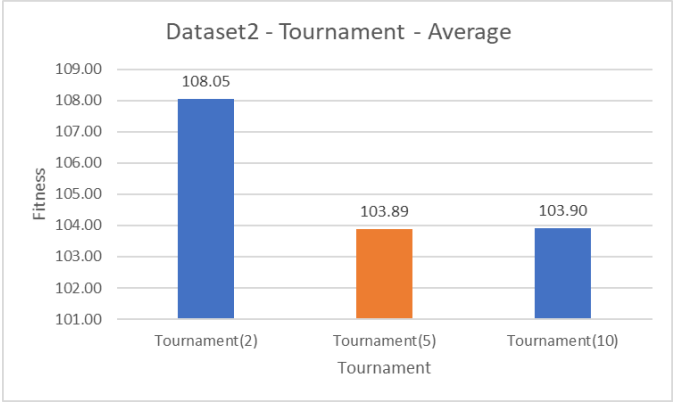
```
Selection: tournament

Tournament_size: 2, 5, 10
```

3.2.3.1 *DataSet1*



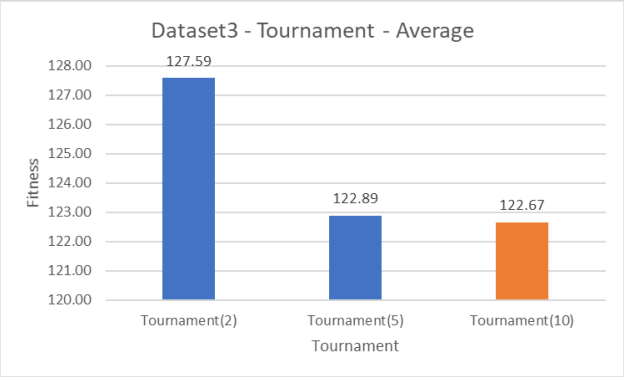
Analisando o gráfico verificamos que o valor 10 é o tamanho do torneio que apresenta os melhores.

3.2.3.2 *DataSet2*



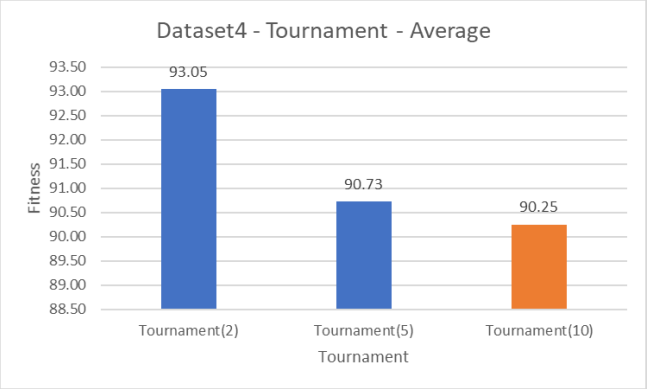
Analisando o gráfico verificamos que o valor 5 é o tamanho do torneio que apresenta os melhores.

3.2.3.3 *DataSet3*



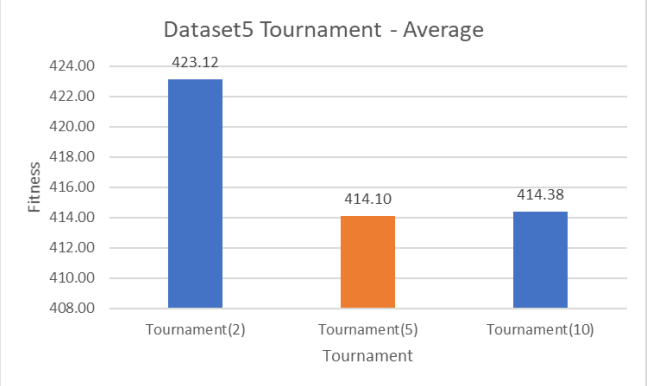
Analisando o gráfico verificamos que o valor 10 é o tamanho do torneio que apresenta os melhores.

3.2.3.4 *DataSet4*



Analisando o gráfico verificamos que o valor 10 é o tamanho do torneio que apresenta os melhores.

3.2.3.5 *DataSet5*



Analisando o gráfico verificamos que o valor 5 é o tamanho do torneio que apresenta os melhores.

Após a realização das experiências neste conjunto de parâmetros, inferimos que o valor do tamanho do torneio que apresenta melhores resultados oscila entre o 5 e o 10, sendo que ambos apresentam resultados extremamente chegados em todos os *dataSets*.

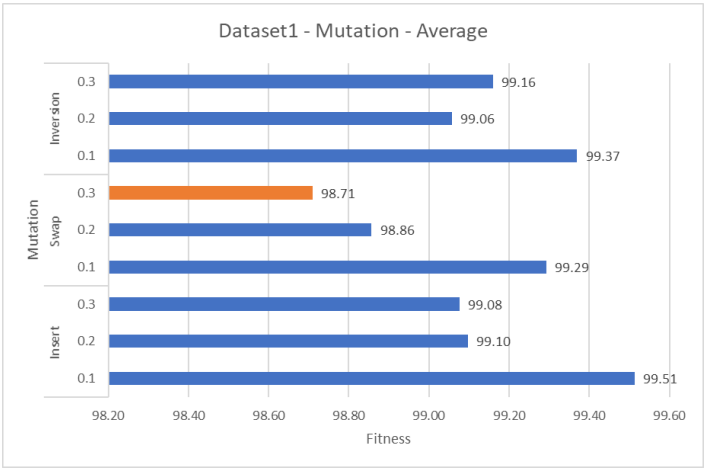
3.2.4 Mutação

Com estas experiências pretendemos analisar o efeito dos operadores de mutação e das respetivas probabilidades de ocorrência.

Para fabricar os testes da mutação, foram inseridos no ficheiro de configurações os seguintes parâmetros:

```
Mutation: insert, swap, inversion

Mutation_probability: 0.1, 0.2, 0.3
```

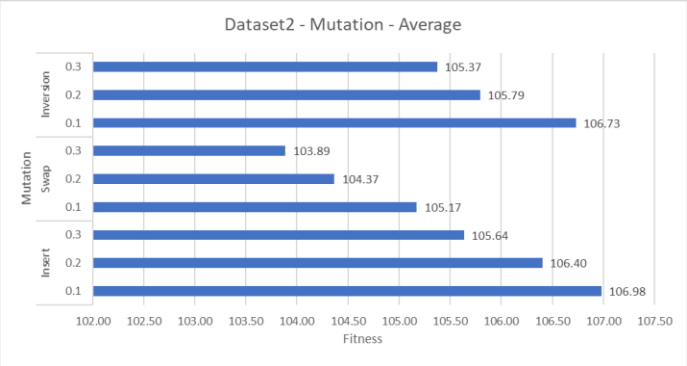
3.2.4.1 *DataSet1*



Através da análise do gráfico podemos concluir que:

- Para a mutação *inversion* a probabilidade que apresenta os melhores resultados é a 0.2
- Para a mutação *swap* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *insert* a probabilidade que apresenta os melhores resultados é a 0.3
- A nível global a melhor combinação de parâmetros é a mutação *swap* aliada à probabilidade 0.3
- A nível global a pior combinação de parâmetros é a mutação *insert* aliada à probabilidade 0.1

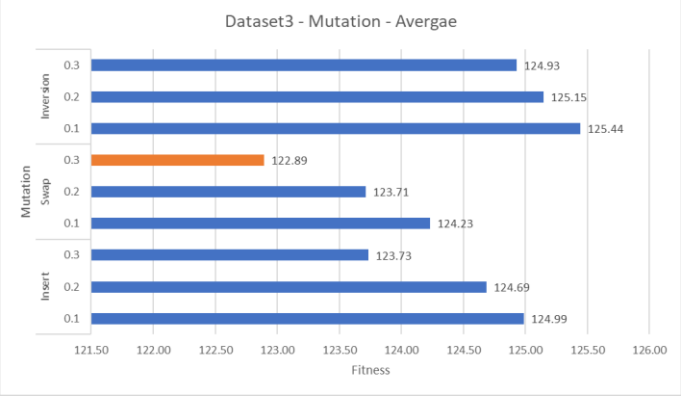
3.2.4.2 *DataSet2*



Através da análise do gráfico podemos concluir que:

- Para a mutação *inversion* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *swap* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *insert* a probabilidade que apresenta os melhores resultados é a 0.3
- A nível global a melhor combinação de parâmetros é a mutação *swap* aliada à probabilidade 0.3
- A nível global a pior combinação de parâmetros é a mutação *insert* aliada à probabilidade 0.1

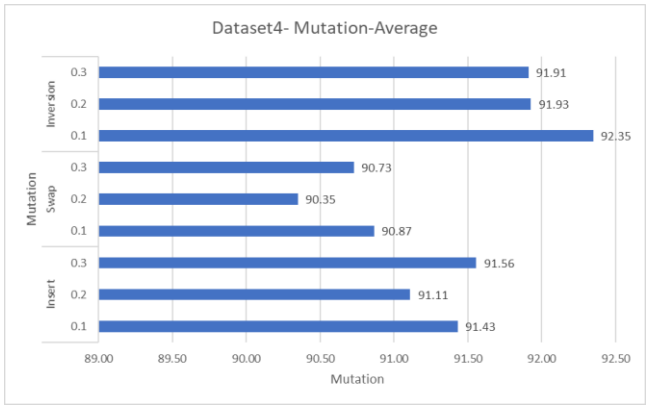
3.2.4.3 *DataSet3*



Através da análise do gráfico podemos concluir que:

- Para a mutação *inversion* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *swap* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *insert* a probabilidade que apresenta os melhores resultados é a 0.3
- A nível global a melhor combinação de parâmetros é a mutação *swap* aliada à probabilidade 0.3
- A nível global a pior combinação de parâmetros é a mutação *inversion* aliada à probabilidade 0.1

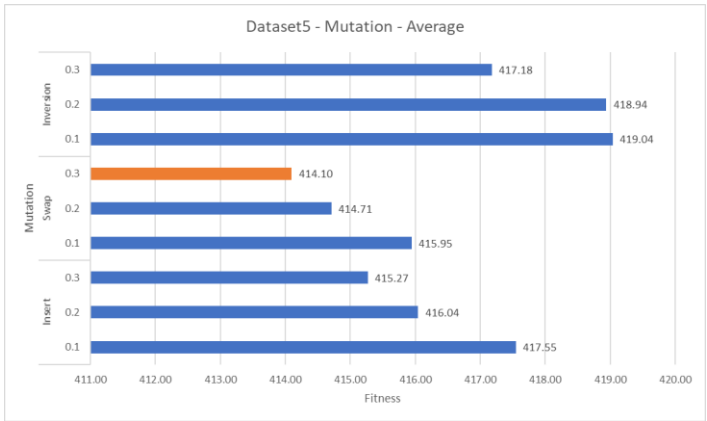
3.2.4.4 *DataSet4*



Através da análise do gráfico podemos concluir que:

- Para a mutação *inversion* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *swap* a probabilidade que apresenta os melhores resultados é a 0.2
- Para a mutação *insert* a probabilidade que apresenta os melhores resultados é a 0.2
- A nível global a melhor combinação de parâmetros é a mutação *swap* aliada à probabilidade 0.2
- A nível global a pior combinação de parâmetros é a mutação *inversion* aliada à probabilidade 0.1

3.2.4.5 *DataSet5*



Através da análise do gráfico podemos concluir que:

- Para a mutação *inversion* a probabilidade que apresenta os melhores resultados é a 0.3
- Para a mutação *swap* a probabilidade que apresenta os melhores resultados é a 0.3

- Para a mutação *insert* a probabilidade que apresenta os melhores resultados é a 0.3
- A nível global a melhor combinação de parâmetros é a mutação *swap* aliada à probabilidade 0.3
- A nível global a pior combinação de parâmetros é a mutação *inversion* aliada à probabilidade 0.1

Podemos concluir que a mutação do tipo *swap* com a probabilidade 0.3 apresentou em média o melhor fitness perante todos testes realizados nos *datasets*.

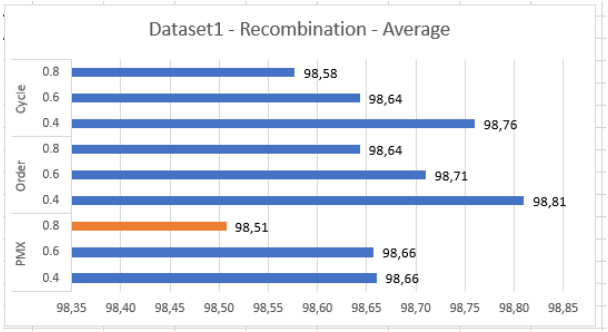
3.2.5 Recombinação

Com estas experiências pretendemos analisar o efeito dos operadores de recombinação e das respetivas probabilidades de ocorrência.

Para fabricar os testes da recombinação, foram inseridos no ficheiro de configurações os seguintes parâmetros:

```
Recombination: pmx, order, cycle

Recombination_probability: 0.4, 0.6, 0.8
```

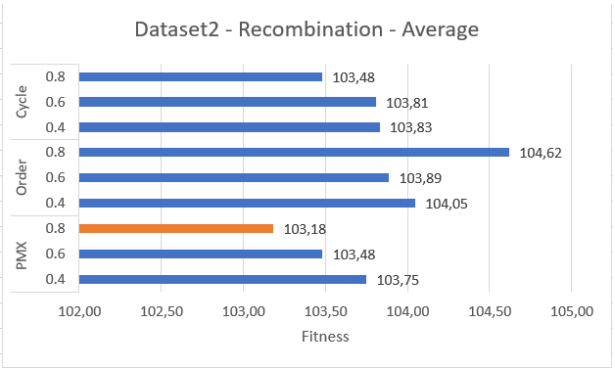
3.2.5.1 Dataset1



Através da análise do gráfico podemos concluir que:

- Para a recombinação *cycle* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *order* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *PMX* a probabilidade que apresenta os melhores resultados é a 0.8
- A nível global a melhor combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.8
- A nível global a pior combinação de parâmetros é a recombinação *order* aliada à probabilidade 0.4

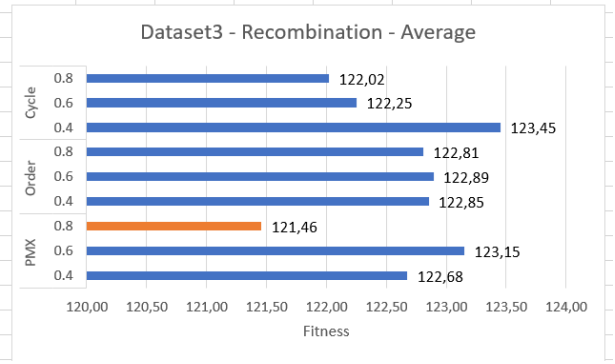
3.2.5.2 Dataset2



Através da análise do gráfico podemos concluir que:

- Para a recombinação *cycle* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *order* a probabilidade que apresenta os melhores resultados é a 0.6
- Para a recombinação *PMX* a probabilidade que apresenta os melhores resultados é a 0.8
- A nível global a melhor combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.8
- A nível global a pior combinação de parâmetros é a recombinação *order* aliada à probabilidade 0.8

3.2.5.3 Dataset3

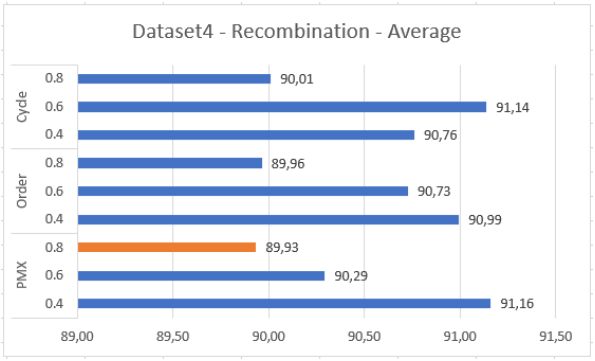


Através da análise do gráfico podemos concluir que:

- Para a recombinação *cycle* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *order* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *PMX* a probabilidade que apresenta os melhores resultados é a 0.8
- A nível global a melhor combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.8

- A nível global a pior combinação de parâmetros é a recombinação *cycle* aliada à probabilidade 0.4

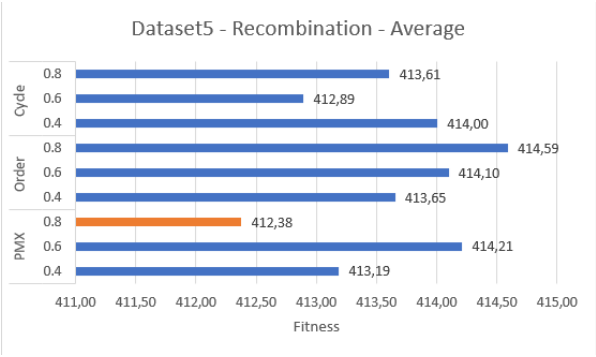
3.2.5.4 *Dataset4*



Através da análise do gráfico podemos concluir que:

- Para a recombinação *cycle* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *order* a probabilidade que apresenta os melhores resultados é a 0.8
- Para a recombinação *PMX* a probabilidade que apresenta os melhores resultados é a 0.8
- A nível global a melhor combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.8
- A nível global a pior combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.4

3.2.5.5 *Dataset5*



Através da análise do gráfico podemos concluir que:

- Para a recombinação *cycle* a probabilidade que apresenta os melhores resultados é a 0.6
- Para a recombinação *order* a probabilidade que apresenta os melhores resultados é a 0.4
- Para a recombinação *PMX* a probabilidade que apresenta os melhores resultados é a 0.8
- A nível global a melhor combinação de parâmetros é a recombinação *PMX* aliada à probabilidade 0.8
- A nível global a pior combinação de parâmetros é a recombinação *order* aliada à probabilidade 0.8

Podemos concluir que a recombinação do tipo *PMX* com a probabilidade 0.8 apresentou em média o melhor fitness perante todos testes realizados nos *datasets*.

3.2.6 Testes Gerais

Num contexto ideal os testes gerais eram as primeiras experiências a ser realizadas, para se poder definir com exatidão o ficheiro *config*. que iria ser utilizado pela aplicação para efetuar os múltiplos testes com várias combinações de parâmetros. Mas como aquando do início do processo experimental ainda não tínhamos concluído a implementação da recombinação *cycle*, tivemos de optar por definir os parâmetros do ficheiro *config* através de um processo experimental baseado na especulação.

Dito isto, o objetivo dos testes realizados neste capítulo é averiguar qual é o conjunto de parâmetros que apresenta melhores resultados para cada *dataSet* e construir uma tabela com base nesses dados.

Para fabricar os testes gerais, foram inseridos no ficheiro de configurações os seguintes parâmetros:

```
Selection: tournament

Tournament_size: 2, 5, 10

//--------------------------------

Recombination: pmx, order, cycle

Recombination_probability: 0.4, 0.6, 0.8

//--------------------------------

Mutation: insert, swap, inversion

Mutation_probability: 0.1, 0.2, 0.3
```

Visto que, a população e a geração apresentam uma relação de proporcionalidade, formam realizados testes tendo em conta 3 relações de proporcionalidade população/geração diferentes: 50/200, 200/50 e 400/25.

Melhor combinação de parâmetros para cada dataSet

dataSet	população/ gerações	size torneio	recomb.	mutação	fitness
1	200/50	2	Cycle (0.8)	Insert (0.3)	98.40
1	200/50	2	Cycle (0.8)	Swap (0.3)	98.40
2	200/50	5	PMX (0.8)	Swap (0.3)	103.18
3	50/200	2	Cycle (0.8)	Inversion (0.3)	121.21
4	200/50	2	Cycle (0.6)	Swap (0.3)	89.30
5	200/50	5	PMX (0.8)	Swap (0.3)	412.38

De notar que, para os *dataSets* 2 e 4 apenas foram realizados testes para a relação de proporcionalidade população/geração de 200/50. Os valores utilizados para construir a tabela foram retiradas dos respetivos ficheiros excel (*statistic_average_fitness.xls*).

Dos valores correspondentes ao fitness recolhidos após processo experimental, podemos inferir que aliados à variação de outros parâmetros:

- a relação população/geração de 200/50 foi a que mais contribuiu para obtenção do melhor fitness;
- o tamanho de torneio 10 não contribuiu para obtenção do melhor fitness em nenhum *dataSet;*
- a recombinação *Order* não contribuiu para obtenção do melhor fitness em nenhum *dataSet;*
- a probabilidade de recombinação 0.8 contribuiu no geral para a obtenção do melhor resultado do fitness;
- a mutação *Swap* foi a que mais contribuiu para obtenção do melhor fitness;
- a probabilidade de mutação 0.3 contribuiu universalmente para a obtenção do melhor resultado do fitness.

4. OBSERVAÇÕES

Por uma razão que não conseguimos apurar, o valor do excel best fitness (statistic_best_per_experiment_fitness), não coincide com valor do melhor fitness do excel average fitness (statistic_average_fitness). Sendo que esse aspeto não interferiu com processo experimental.

5. CONCLUSÃO

Dado por concluído o projeto proposto, podemos afirmar que os componentes cruciais para o bom funcionamento da aplicação foram implementados com sucesso. Sendo prova disso, o facto de os resultados obtidos no processo experimental corresponderem aos valores espectados, mesmo tendo em conta que enfrentámos alguns problemas inicias na execução dos testes.

6. REFERÊNCIAS

[1] https://www.baeldung.com/java-genetic-algorithm

[2] https://www.youtube.com/watch?v=FKhgrb2zaMA

[3] https://stackabuse.com/introduction-to-genetic-algorithms-in-java

[4] https://www.youtube.com/watch?v=HATPHZ6P7c4

[5] https://www.youtube.com/watch?v=4YjNe3qvVlI&list=RDCMUCNYv4HA3WjV3gZGLfBehRWQ&start_radio=1

[6] https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

[7] https://github.com/PLT875/Solving-the-TSP-using-Genetic-Algorithms/blob/master/src/Crossover/OX.java

[8] https://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/CycleCrossoverOperator.aspx

[9] https://setu677.medium.com/how-to-perform-roulette-wheel-and-rank-based-selection-in-a-genetic-algorithm-d0829a37a189