Otimização de Espaço

Projeto realizado para a unidade curricular de Inteligência Artificial do curso de Engenharia Informática do Politécnico de Leiria para o ano letivo 2020/2021.

Ana Cassia Cruz Vasconcelos
DEI
IPLeiria - ESTG
Leiria, Portugal
2201736@my.ipleiria.pt

Enrico Florentino Gomes
DEI
IPLeiria - ESTG
Leiria, Portugal
2181895@my.ipleiria.pt

Resumo

Este documento tem como objetivo descrever a nossa solução relativa ao projeto proposto na componente prática da Unidade Curricular de Inteligência Artificial. O objetivo do trabalho realizado é apresentar soluções otimizadas para a realização de cortes de peças numa superfície através de uma aplicação desenvolvida e realização de experimentos.

Categorias e descrição do assunto

[**Linguagem de programação**]: Java – *algoritmos genéticos: métodos de recombinação, seleção e mutação; interface gráfica do utilizador (GUI); geração de ficheiro Excel.*

Termos gerais

Algoritmos, Medições, *Performance*, Confiabilidade, Experimentação, Verificações.

Palavras-chaves

Aplicação, Inteligência Artificial, Algoritmos Genéticos, *Java*, *Datasets*, Operadores Genéticos, Recombinação, Mutação, População, Seleção, Torneio.

1. Introdução

Foi proposto para os estudantes da unidade curricular de Inteligência Artificial o desenvolvimento de uma aplicação cuja função é otimizar o posicionamento de peças, posteriormente cortadas, num espaço representado por uma superfície, onde tal posicionamento leve em consideração o menor número de cortes possíveis a serem feitos de forma que seja resolvido o problema.

A aplicação levou no seu desenvolvimento a utilização de algoritmos genéticos e *datasets*, com realização posterior de testes com diversos parâmetros diferentes para validar as melhores combinações possíveis.

Este tipo de aplicação surge no contexto de indústrias onde cortes de peças são realizados sobre superfícies de matéria-prima, onde estas podem ser do ramo têxtil, metalomecânicas, construção civil, entre outras. No caso da aplicação em questão, o contexto insere-se no ramo têxtil.

2. Metodologia de desenvolvimento

A abordagem para a implementação do projeto foi feita utilizando a linguagem de programação *Java* e o ambiente de desenvolvimento *IntelliJ*, onde nestes foram desenvolvidos classes, operadores e algoritmos pertinentes ao problema proposto.

2.1 Classes implementadas

Em relação às classes desenvolvidas, temos:

- A classe ***stockingproblem.StockingProblem***:

  Esta classe se refere ao problema a ser resolvido e é composta pelos parâmetros referentes à altura da superfície a ser cortada (variável *materialHeight*, representada por um número inteiro), à largura da superfície a ser cortada (variável *materialWidth*, representada por um número inteiro) e a distribuição das peças nesta superfície (variável *items*, representada por um A*rrayList* de objetos da classe *Item*, que por sua vez simboliza as peças que devem ser produzidas e é representada por caracteres de 'A' a 'Z', em letras a minúsculo ou a maiúsculo).

  Com a definição da altura e largura (esta sendo definida pela quantidade de peças do *dataset* e seus comprimentos), a classe procede a dividir a superfície a ser cortada por linhas e colunas, de forma a criar uma matriz que abrigue as peças do problema em questão.

- A classe ***stockingproblem.StockingProblemIndividual***:

  Cada indivíduo da população referente ao problema é representado e criado por esta classe, que é composta pelos parâmetros referentes à quantidade de cortes a serem feitos (variável inteira quantidadeCortes), uma matriz de números inteiros que representa o material da superfície (variável material) e o número de colunas deste material (variável inteira tamanhoColunas, inicializada a 0). Estes indíviduos são representados pelo genoma que possuem e as peças que estes genomas contêm.

  A função principal desta classe é a de calcular o *fitness* (função *public double computeFitness()*), responsável não só por calcular o *fitness*, mas também como percorrer a matriz do material do genoma, pela validação do posicionamento das peças, assim como posicionar as mesmas após dita validação.

  O posicionamento das peças do *dataset* no material é realizado pelo percorrer da matriz primeiramente em sentido de cima

para baixo nas colunas e em seguida no sentido da esquerda para a direita nas linhas, com foco em otimizar o espaço ocupado por estas no material, manter todas as peças dentro do espaço deste e sem haver sobreposição das mesmas.

Ao fim da resolução do problema, a função *computeFitness()* retorna o número de cortes necessários de forma otimizada e um *fitness* calculado de acordo com o tamanho de colunas multiplicado por 0,7 somado com 0,3 multiplicado pela altura do material somada à quantidade de cortes necessários. O *fitness* calculado representa a eficiência do processo realizado, de forma que quanto menor o seu valor, mais otimizada foi a solução.

2.2 Operadores genéticos desenvolvidos

Sobre os operadores genéticos desenvolvidos para a aplicação, temos para os que se referem aos operadores de recombinação:

- O método de recombinação cíclica (classe ***ga.geneticoperators.RecombinationCycle***):

  Esta classe recebe genomas de dois indivíduos (*ind1* e *ind2*) e é definida pela probabilidade de recombinação entre eles, produzindo dois filhos (*child1* e *child2*, vetores de inteiros) como resultado da combinação realizada.

  A recombinação cíclica, como o nome sugere, preenche-se em ciclos alternados os alelos dos filhos resultantes. Os filhos são vetores de tamanho equivalente ao número de genes do primeiro pai (*ind1*) e têm índices preenchidos de forma reversa, ou seja, quando se trata de um ciclo ímpar (paridade controlada pela variável inteira *var7*) preenche-se os alelos do primeiro filho (*child1*) com o genoma do segundo pai (*ind2*), e quando é um ciclo par preenche-se os alelos do segundo filho (*child2*) com o genoma do primeiro pai.

  Durante a alternação entre ciclos garante-se que o índice do filho a ser preenchido tem a mesma posição do índice que foi preenchido no filho anterior, ou a seguinte caso esta já tenha sido preenchida.

- O método de recombinação ordenada (classe ***ga.geneticoperators.RecombinationOrdered***):

  Esta classe, como a anterior, também recebe o genoma de dois indivíduos (*ind1* e *ind2*) e é definida pela probabilidade de recombinação ocorrer entre eles, para assim produzir dois filhos (*child1* e *child2*, vetores de inteiros).

  A recombinação ordenada, funciona de forma que os alelos dos filhos sejam preenchidos conservando uma certa ordem dos indivíduos pais. Através das variáveis de controle da posição do corte dos genomas dos pais (*cut1* e *cut2*, números inteiros), uma posição para iniciar o corte do genoma é escolhida de forma aleatória. Em seguida, partindo da posição selecionada no índice do genoma do primeiro pai, todos os restantes alelos serão utilizados para preencher o genoma do primeiro filho, e os alelos do segundo pai, partindo da posição do primeiro alelo até onde foi realizado o corte no primeiro pai, são utilizados para preencher o resto do genoma do primeiro filho.

  Esta operação é a realizada de forma inversa, ou seja o segundo pai dando início à operação ao invés do primeiro, para preencher o genoma do segundo filho.

Já sobre os operadores de mutação desenvolvidos, temos:

- O método de mutação por inversão (classe ***ga.geneticoperators.MutationInversion***):

  Esta classe recebe o indivíduo (*ind*) que sofrerá a mutação e é definida pela probabilidade da mutação ocorrer, procedendo por aleatoriamente escolher uma posição para iniciar a inversão do genoma deste. Após a posição de corte do genoma ser definida randomicamente através das variáveis de controle de corte (*cut1* e *cut2*), inverte-se todos os alelos a partir da posição escolhida, de forma que os alelos a partir desta são reescritos na ordem inversa, conservando os restantes.

- O método de mutação por troca de posição (classe ***ga.geneticoperators.MutationSwap***):

  Esta classe pode ser considerada a mais simples das desenvolvidas, recebendo apenas um indivíduo (*ind*) e é definida pela probabilidade da mutação para este ocorrer, de forma que quando esta ocorre, escolhe duas posições de forma aleatória para trocar os alelos entre elas.

3. Realização de experimentos

Para cada *dataset* foi realizado testes gerais com valores relevantes para os experimentos. Inicialmente fizemos para a população a 50 e a 100 com 50 gerações e posteriormente fizemos para população a 200 também com 50 gerações separadamente. Só achamos relevante adicionar a população 200 quando nos testes da população o melhor resultado obtido foi com a população 200 e geração a 50, que contradizia com o resultado dos testes gerais que inicialmente dava população a 100 e geração a 50 como a melhor com uma diferença relevante de valores.

Os restantes parâmetros dos testes gerais foram:

- Torneio: 2, 4, 6 e 8
- Recombinações pmx, cycle e order: 0.5, 0.6, 0.7, 0.8, 0.9
- Mutações insert, swap e invert: 0.01, 0.025, 0.3

Na escolha dos valores para os parâmetros, foi feito testes para cada problema, utilizando diversos valores para perceber qual mantia a reta azul (*average*) do gráfico da interface o mais estável possível, ou seja, o mais próximo possível da reta vermelha (*best*). Assim, se o problema estabilizava com uma recombinação de 0.6 colocamos os valores a partir do 0.5, ou seja, começamos com um valor abaixo do que aparentemente seria o melhor resultado, para se ter a certeza.

As mutações possuem valores muito pequenos como 0.01, 0.025 e 0.03, a recombinação costuma possuir valores mais elevados, a partir de 0.05. Por isto, nas mutações colocamos valores mais baixos para os testes gerais.

Na próxima secção vamos falar de forma mais aprofundada sobre cada teste de cada *dataset* para avaliarmos se os resultados são correspondentes ou não.

4. Discussão dos resultados

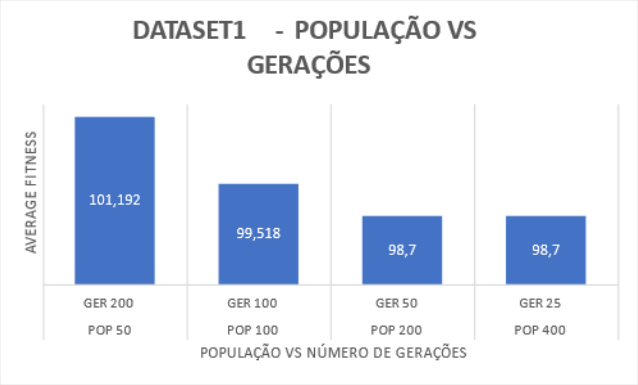
4.1.1 DataSet1

Para o teste geral deste *dataset*, o melhor resultado foi: população a 200, geração a 50, torneio a 2, recombinação *cycle* a 0.8,

mutação *insert* a 0.01 e *average fitness* com resultado de 98,7. Este melhor resultado foi encontrado após ordenar de forma crescente o *average fitness*, apesar de "empatar" com mais resultados levamos em conta o que aparecia na primeira linha com o melhor resultado.

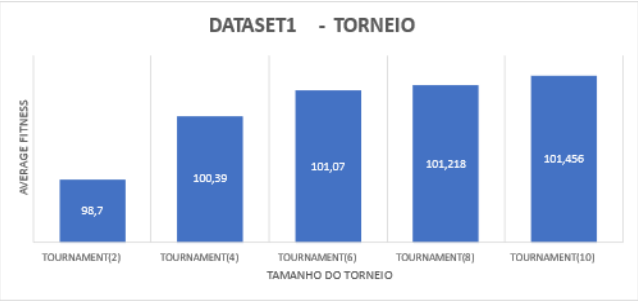
População: o *config.txt* para este teste teve os seguintes valores para a população: 50, 100, 200, 400; com valores de geração: 25, 50, 100 e 200; os outros parâmetros foram iguais aos do melhor resultado dos testes gerais.

O gráfico a seguir mostra que de fato a população a 200 e geração a 50 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor população e geração, mas também tendo empate com população a 400 e geração a 50. Como nos testes gerais não havia população com valor a 400, provavelmente os valores poderiam "empatar" como melhores resultados para população.



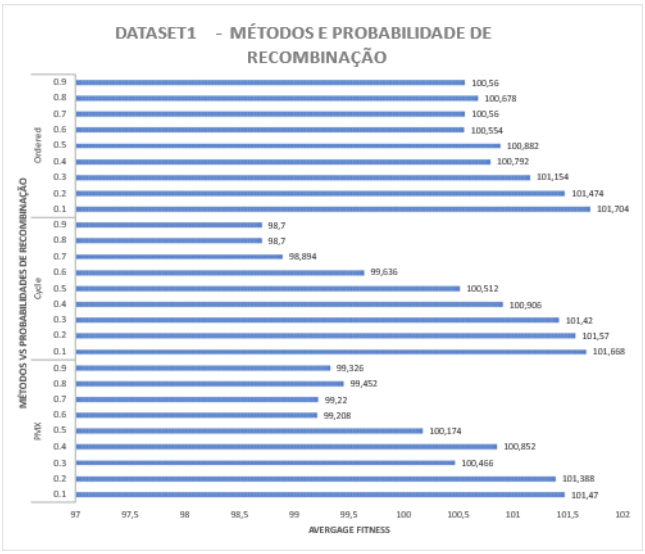
Torneio: o *config.txt* para este teste teve os seguintes valores no parâmetro torneio: 2, 4, 6, 8 e 10; com os outros parametros tendo valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o torneio com tamanho 2 possui um *average fitness* mais baixo, correspondendo ao resultado do teste geral como melhor torneio.



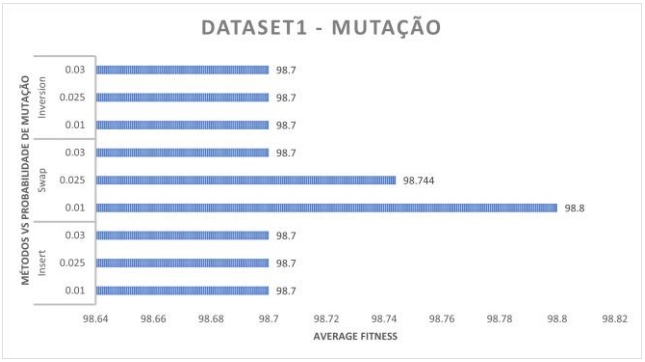
Recombinação: no *config.txt* a recombinação teve *pmx*, *cycle* e *order* habilitados com probabilidades a: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 e 0.9; tendo os outros parâmetros equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o *cycle* com valor a 0.8 possui um *average fitness* mais baixo, correspondendo ao resultado do teste geral como melhor recombinação, mas ao mesmo tempo tendo empate com o *cycle* a 0.9. Nos testes gerais, o *cycle* a 0.9 só aparece muito abaixo da primeira linha.



Mutação: o *config.txt* teve mutação habilitada para *insert*, *swap* e *inverse* para este teste com probabilidades a: 0.01, 0.025 e 0.03, mantendo os parâmetros com mesmos valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que o parâmetro *mutation* tem pior eficiência quando se trata do tipo *swap* e probabilidade diferente de 0.03, desta forma confirmando o melhor resultado do teste geral com mutação *insert* a 0.01. Os restantes parâmetros resultam em fitness equivalente.

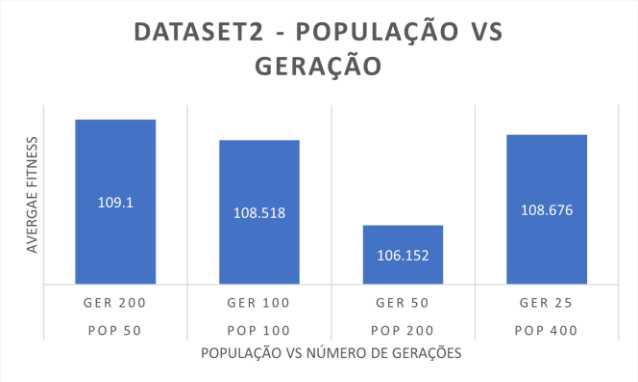


4.1.2 DataSet2

Para o teste geral deste *dataset*, o melhor resultado foi: população a 200, geração a 50, torneio a 2, recombinação *cycle* a 0.9, mutação *insert* a 0.03 e *average fitness* com resultado de 106,152. Este melhor resultado foi encontrado após ordenar de forma crescente o *average fitness*.

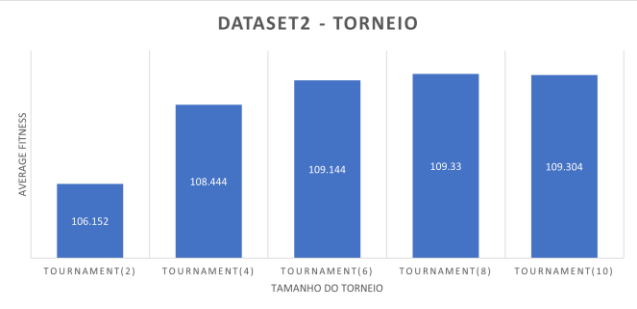
População: o *config.txt* para este teste teve os seguintes valores para a população: 50, 100, 200, 400; com valores de geração: 25, 50, 100 e 200; os outros parâmetros foram iguais aos do melhor resultado dos testes gerais.

O gráfico a seguir confirma que a população com valor a 200 e geração com valor a 50 possui o menor *average fitness*, o que se encaixa no contexo do melhor resultado dos testes gerais.



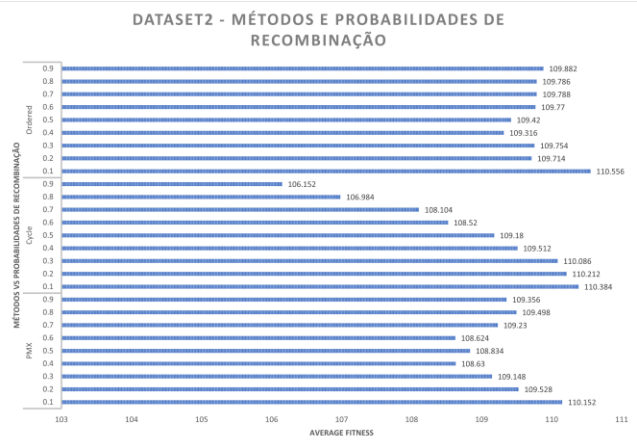
Torneio: o *config.txt* para este teste teve os seguintes valores no parâmetro torneio: 2, 4, 6, 8 e 10; com os outros parametros tendo valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o torneio com tamanho a 2 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor torneio.



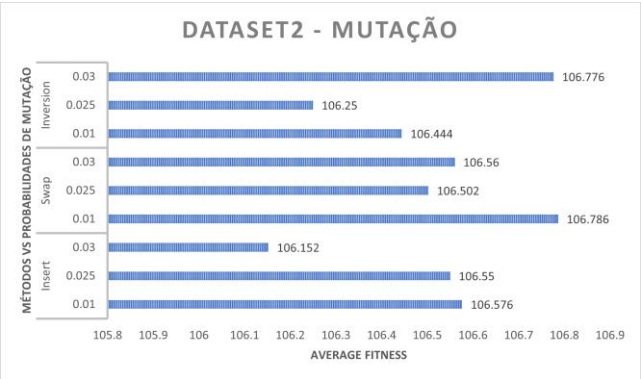
Recombinação: no *config.txt* a recombinação teve *pmx*, *cycle* e *order* habilitados com probabilidades a: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 e 0.9; tendo os outros parâmetros equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o *cycle* com valor a 0.9 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor recombinação.



Mutação: o *config.txt* teve mutação habilitada para *insert*, *swap* e *inverse* para este teste com probabilidades a: 0.01, 0.025 e 0.03, mantendo os parâmetros com mesmos valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o parâmetro *mutation* tem melhor eficiência quando se trata do tipo *insert* e probabilidade de 0.03.

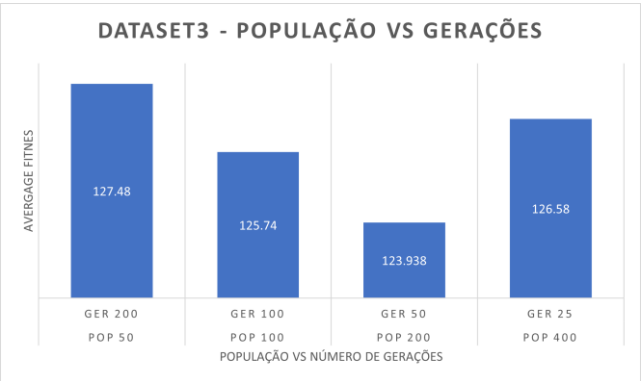


4.1.3 DataSet3

Para o teste geral deste *dataset*, o melhor resultado foi: população a 200, geração a 50, torneio a 4, recombinação *pmx* a 0.9, mutação *insert* a 0.03 e *average fitness* com resultado de 123,938. Este melhor resultado foi encontrado após ordenar de forma crescente o *average fitness*.

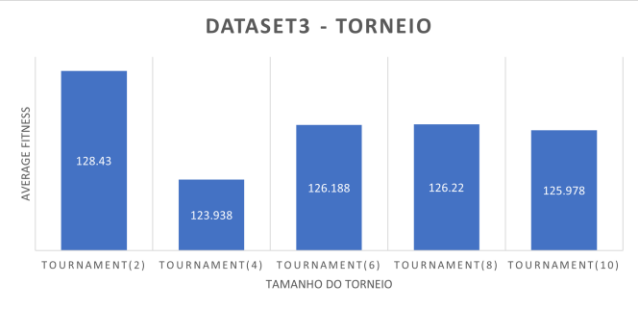
População: o *config.txt* para este teste teve os seguintes valores para a população: 50, 100, 200, 400; com valores de geração: 25, 50, 100 e 200; os outros parâmetros foram iguais aos do melhor resultado dos testes gerais.

O gráfico a seguir confirma que a população com valor a 200 e geração com valor a 50 possui o menor *average fitness*, o que se encaixa no contexo do melhor resultado dos testes gerais.



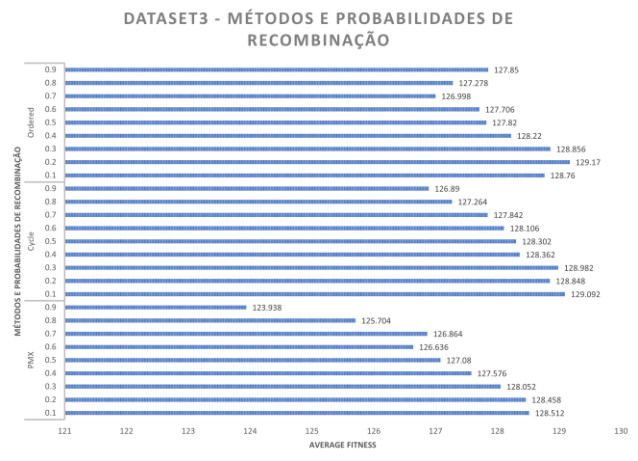
Torneio: o *config.txt* para este teste teve os seguintes valores no parâmetro torneio: 2, 4, 6, 8 e 10; com os outros parametros tendo valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o torneio com tamanho a 4 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor torneio.



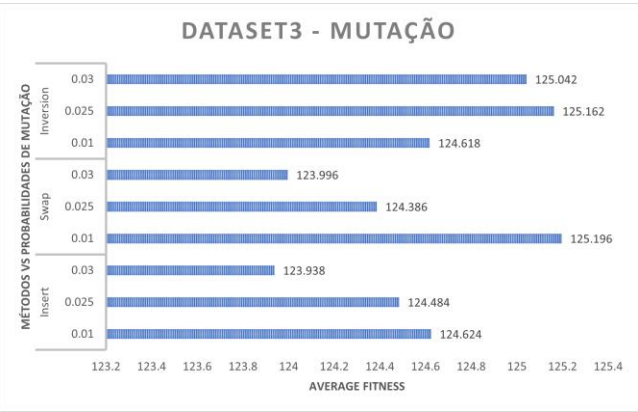
Recombinação: no *config.txt* a recombinação teve *pmx*, *cycle* e *order* habilitados com probabilidades a: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 e 0.9; tendo os outros parâmetros equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o *pmx* com valor a 0.9 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor recombinação.



Mutação: o *config.txt* teve mutação habilitada para *insert*, *swap* e *inverse* para este teste com probabilidades a: 0.01, 0.025 e 0.03, mantendo os parâmetros com mesmos valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o parâmetro *mutation* tem melhor eficiência quando se trata do tipo *insert* e probabilidade de 0.03.

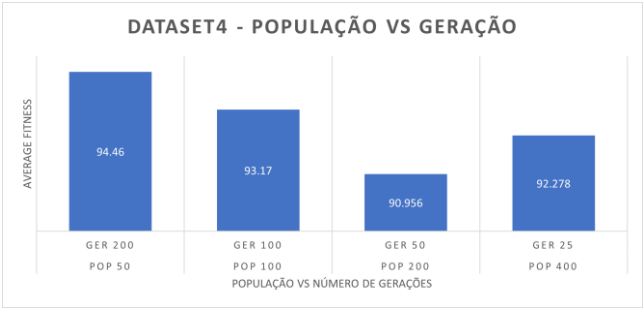


4.1.4 DataSet4

Para o teste geral deste *dataset*, o melhor resultado foi: população a 200, geração a 50, torneio a 2, recombinação *cycle* a 0.9, mutação *insert* a 0.025 e *average fitness* com resultado de 90,956. Este melhor resultado foi encontrado após ordenar de forma crescente o *average fitness*.

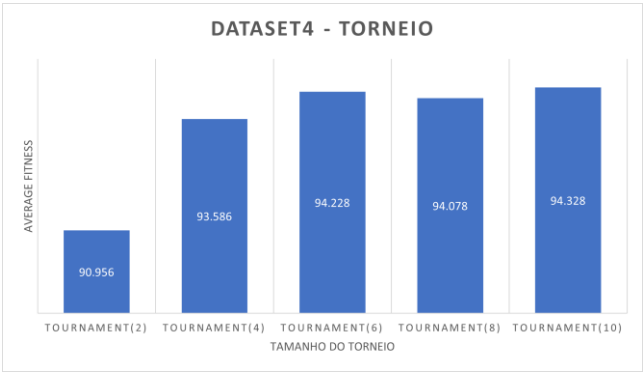
População: o *config.txt* para este teste teve os seguintes valores para a população: 50, 100, 200, 400; com valores de geração: 25, 50, 100 e 200; os outros parâmetros foram iguais aos do melhor resultado dos testes gerais.

O gráfico a seguir confirma que a população com valor a 200 e geração com valor a 50 possui o menor *average fitness*, o que se encaixa no contexo do melhor resultado dos testes gerais.



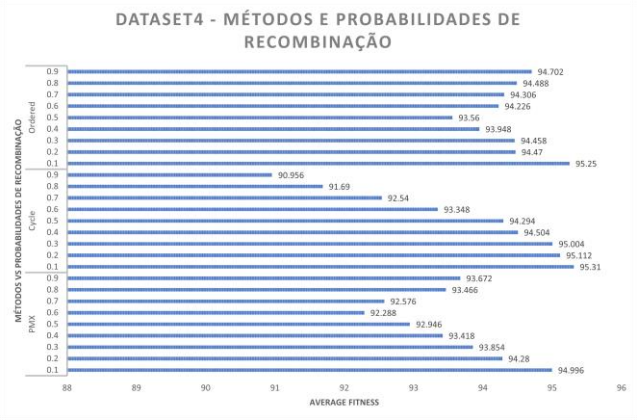
Torneio: o *config.txt* para este teste teve os seguintes valores no parâmetro torneio: 2, 4, 6, 8 e 10; com os outros parametros tendo valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o torneio com tamanho a 2 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor torneio.



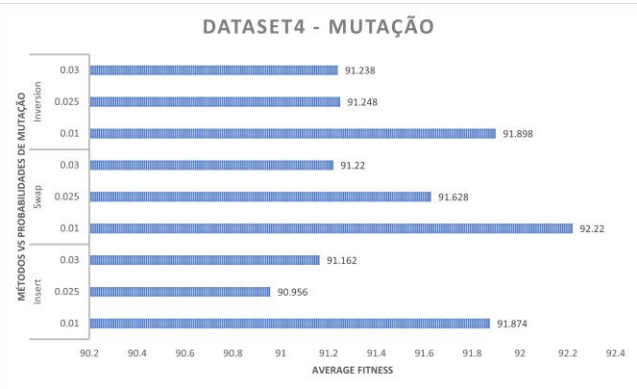
Recombinação: no *config.txt* a recombinação teve *pmx*, *cycle* e *order* habilitados com probabilidades a: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 e 0.9; tendo os outros parâmetros equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o *cycle* com valor a 0.9 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor recombinação.



Mutação: o *config.txt* teve mutação habilitada para *insert*, *swap* e *inverse* para este teste com probabilidades a: 0.01, 0.025 e 0.03, mantendo os parâmetros com mesmos valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o parâmetro *mutation* tem melhor eficiência quando se trata do tipo *insert* e probabilidade de 0.025.

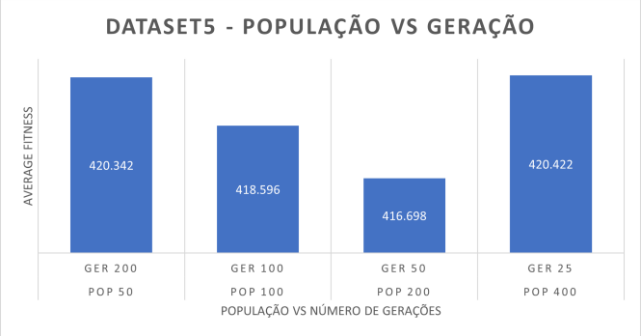


4.1.5 DataSet5

Para o teste geral deste *dataset*, o melhor resultado foi: população a 200, geração a 50, torneio a 4, recombinação *pmx* a 0.9, mutação *insert* a 0.03 e *average fitness* com resultado de 416,698. Este melhor resultado foi encontrado após ordenar de forma crescente o *average fitness*.

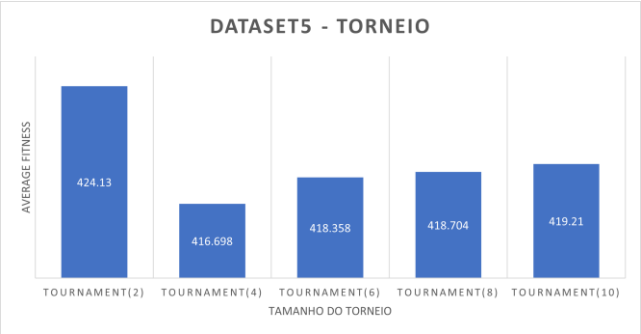
População: o *config.txt* para este teste teve os seguintes valores para a população: 50, 100, 200, 400; com valores de geração: 25, 50, 100 e 200; os outros parâmetros foram iguais aos do melhor resultado dos testes gerais.

O gráfico a seguir confirma que a população com valor a 200 e geração com valor a 50 possui o menor *average fitness*, o que se encaixa no contexo do melhor resultado dos testes gerais.



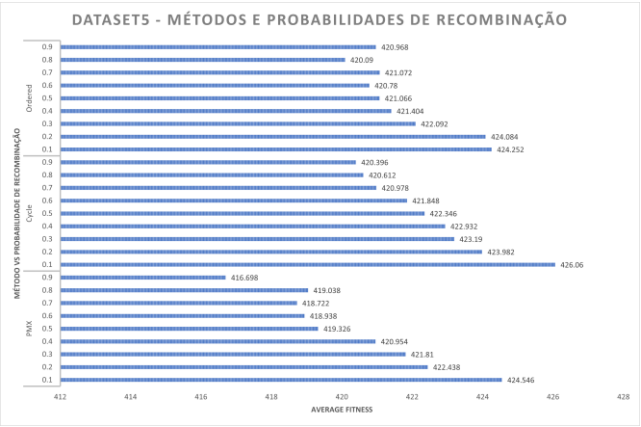
Torneio: o *config.txt* para este teste teve os seguintes valores no parâmetro torneio: 2, 4, 6, 8 e 10; com os outros parametros tendo valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o torneio com tamanho a 4 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor torneio.



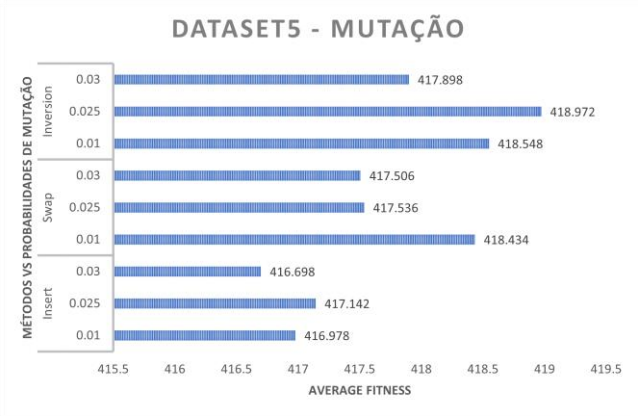
Recombinação: no *config.txt* a recombinação teve *pmx*, *cycle* e *order* habilitados com probabilidades a: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 e 0.9; tendo os outros parâmetros equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir mostra que de fato o *pmx* com valor a 0.9 possui o menor *average fitness*, correspondendo ao resultado do teste geral como melhor recombinação.



Mutação: o *config.txt* teve mutação habilitada para *insert*, *swap* e *inverse* para este teste com probabilidades a: 0.01, 0.025 e 0.03, mantendo os parâmetros com mesmos valores equivalentes aos do melhor resultado nos testes gerais.

O gráfico a seguir confirma que o parâmetro *mutation* tem melhor eficiência quando se trata do tipo *insert* e probabilidade de 0.03.



5. Referências

[1] *Artificial Intelligence: A Modern Approach*, Stuart Russel, Peter Norvig, Prentice Hall, 3rd edition, 2009.

[2] *Artificial Intelligence: A New Synthesis*, Nils J. Nilson, Morgan Kaufmann, 1st edition, 1998.

[3] *Inteligência Artificial: Fundamentos e Aplicações*, Ernesto Costa, Anabela Simões, FCA, 2ª edição, 2008.

[4] Apontamentos disponibilizados pelos docentes.