• U  C •

Computação Evolucionária

TP16 – Inserção de Imigrantes

Miguel Faria
2019216809
miguelfaria@student.dei.uc.pt

1 Introdução

A computação evolucionária é uma ferramenta bastante útil e poderosa que possibilita a resolução de diversos tipos de problemas, especialmente os de otimização, onde a solução pode ser muito ampla e de difícil aperfeiçoamento. Este ramo da Inteligência Artificial é baseado em princípios da natureza, como seleção natural e herança genética; deste modo, este tipo de algoritmos procura adaptar essas ideias criando uma população de possíveis soluções que, após sofrer evoluções através de operadores de variação (mutação e cruzamento), irá originar uma solução que se espera ser ótima (ou perto dela), usando uma função de *fitness* na avaliação dessas soluções, permitindo medir o desempenho de cada uma delas.

Um dos principais desafios da computação evolucionária é manter a diversidade na população, uma vez que, sendo heurísticos, estes algoritmos podem evoluir em direção a uma solução que irá ser ótima localmente, não representando a melhor solução possível. Neste caso, pode-se considerar que ocorre convergência prematura: a procura concentra-se num espaço restrito de soluções, sem explorar outras áreas promissoras. Assim, de maneira a manter a diversidade, é importante adotar estratégias que visem combater esse problema, como, por exemplo, a inserção de novos indivíduos (imigrantes), que possibilitam a preservação de heterogeneidade durante o processo de busca, bem como a exploração de possíveis melhores áreas do espaço de soluções, aumentando a probabilidade de encontrar uma solução global melhor.

A inserção de imigrantes consiste na substituição dos piores elementos de uma população por novos que permitam manter os níveis de diversidade elevados, em cada geração. Em relação à escolha dos indivíduos a serem colocados, existem vários métodos, sendo que vão ser alvo de estudo a inserção de indivíduos aleatórios (*random immigrants*) e de mutações do melhor indivíduo (*elitist immigrants*).

Com este trabalho pretende-se verificar se a inclusão de novos indivíduos melhora o desempenho de algoritmos evolucionários, ao analisar a influência exercida na solução obtida para alguns problemas, de diversos domínios e complexidades. Por conseguinte, foram realizados vários testes, comparando as diferentes abordagens e observando as desigualdades nos casos em que este processo é ou não utilizado.

2 Problemas de Referência

Para poder resolver os problemas com recurso a computação evolucionária, é necessário gerar uma representação do problema, de forma a poder ser utilizado no algoritmo. Assim, visto que a escolha dos operadores de variação mais adequados depende dessa representação, foram escolhidos como alvo de estudo problemas com representações binárias e reais.

Para avaliar o desempenho dos algoritmos possuindo diferentes parâmetros, foram reunidos vários problemas de referência, ou *benchmarks*, de maneira que a cada tipo de problema sejam atribuídos 2 *benchmarks* de complexidades distintas, para melhor poder retirar conclusões.

Relativamente aos problemas de representação binária, foram escolhidos *One Max* (*Trap*) e *João Brandão's Numbers*. No que diz respeito aos de representação com reais, optou-se por selecionar duas funções com características distintas, *Quartic* (De Jong F4) e *Rastrigin*: apesar de ambas serem contínuas, têm diferentes modalidades e convexidades, sendo a segunda de maior grau de dificuldade, devido ao elevado número de ótimos locais.

O problema *One* Max consiste em encontrar uma solução de uma *string* binária onde o número de 1s é máximo. *João Brandão's Numbers* é um desafio onde se pretende obter uma *string* binária, indicadora de números, de modo que não exista um número que seja a média de outros dois. Já as *benchmarks* escolhidas para representações reais correspondem a funções contínuas onde se procura encontrar o mínimo global. Algumas características destes problemas estão detalhadas de seguida:

Representação	Binária	
Problema	One Max	João Brandão's Numbers
Domínio	{0, 1}	{0, 1}

Tabela 1 - Características dos problemas de representação binária selecionados

Representação	Real	
Problema	Quartic Function	Rastrigin Function
Domínio	[-1.28; 1.28]	[-5.12, 5.12]
Mínimo	Min(F4) = F4(0, . . . , 0) = 0	Min(R) = R(0, . . . , 0) = 0

Tabela 2 - Características dos problemas de representação real selecionados

3 Implementação

Inicialmente, foi determinada a estrutura do algoritmo evolucionário, tendo como base os códigos *Simple Evolutionary Algorithm* fornecidos pelo professor, que, de uma maneira geral, podem ser representados pelo pseudocódigo presente na *Figura 1* ilustrada abaixo. O algoritmo começa por gerar uma população aleatória, que é sujeita à medição do seu desempenho (*fitness*) inicial. Após isto, para cada geração subsequente, são aplicados os operadores de variação e técnicas de seleção de indivíduos, bem como a atualização do seu desempenho. No final, é esperado que os indivíduos tenham sofrido alterações, de modo que o seu *fitness* melhore, retornando o melhor indivíduo. Importante salientar que cada um dos testes foi executado 30 vezes (*runs*).

Algorithm 1: Simple Evolutionary Algorithm

Input : NumGener,Problem
Output: Best

```
Population ← RandomPopulation(Problem)
Population ← EvalPopulation(Population)
foreach gener_i ∈ NumGener do
    Mates ← SelectParents(Population)
    Offspring ← Variation(Mates)
    Offspring ← EvalPopulation(Offspring)
    Population ← SelectSurvivors(Population, Offspring)
return BestIndividual(Population)
```

Figura 1 - Pseudocódigo relativo à estrutura dos algoritmos evolucionários utilizados

No que diz respeito aos operadores de variação, como mencionado previamente, foram escolhidos tendo em conta as duas representações selecionadas. Devido a questões de tempo e recursos, decidiu-se utilizar apenas um método para cada um dos operadores (mutação e cruzamento), atribuindo, no entanto, operadores adequados para cada um dos cenários. No caso dos problemas com genótipo binário, é usado *Flip Mutation* e *One-Point Crossover*. Para os problemas com genótipo real, decidiu-se aplicar *Uniform Mutation* e *Arithmetical Crossover*. Além disso, o método usado na escolha dos pais (*parent selection)* foi *Tournament Selection* e para a escolha dos sobreviventes (*survival selection*) optou-se por *Elitism*.

4 Cenário Experimental

Primeiramente, começou-se por realizar testes utilizando diferentes parâmetros que permitissem atingir os resultados mais satisfatórios em cada um dos problemas. Assim, foi possível obter bons dados relativos ao uso do algoritmo evolucionário padrão, bem como alcançar uma boa base para a realização das experiências envolvendo a inserção de imigrantes.

Com esse intuito, foram então estabelecidos diversos parâmetros, apresentados na *Tabela 3*, que foram usados na fase de testagem. De notar que foram tidas em atenção todas as combinações e que as *seeds* usadas foram calculadas aleatoriamente, para que não existisse nenhum tipo de favorecimento em nenhum dos testes. Novamente, a fim

de economizar recursos temporais e por ter já sido testado diversas vezes nas aulas, algumas variáveis foram definidas com valores fixos, como é o caso do número de gerações e do tamanho da população. Para estes parâmetros, foram atribuídos valores "relativamente elevados", 1000 e 100 respetivamente, pois: com um número maior de gerações, o algoritmo tem mais oportunidades para explorar o espaço de busca e convergir para soluções melhores; com uma população maior, há uma maior diversidade de soluções, podendo aumentar a probabilidade de encontrar soluções mais apropriadas.

Problemas / **Parâmetro**	One Max João Brandão's Numbers	Quartic Function Rastrigin Function
Individual Size / Dimensions	100	25
Number of Generations	1000	1000
Population Size	100	100
Mutation Method	Flip Mutation	Uniform Mutation
Mutation Probability	0.05, 0.1, 0.15	0.05, 0.1, 0.15
Crossover Method	One-Point Crossover	Arithmetical Crossover (α=0.5)
Crossover Probability	0.7, 0.8, 0.9	0.7, 0.8, 0.9
Parent Selection Method	Tournament Selection	Tournament Selection
Number of Parents	5	5
Survival Selection Method	Elitism	Elitism
Survival Rate	0.2	0.2

Tabela 3 - Parâmetros testados para os respetivos problemas

Após encontrar os melhores parâmetros, realizaram-se os testes com inserção de imigrantes. Como referido anteriormente, foram estudados dois métodos distintos, *random* e *elitist immigrants*, tendo a percentagem de indivíduos a ser substituídos sido fixada em 0.2. No caso de inserção de mutações do melhor indivíduo, o método usado é o mesmo usado em cada tipo de problema, binário ou real, com o melhor valor de probabilidade de mutação encontrado durante a testagem de parâmetros.

Deste modo, foram efetuadas as 3 experiências (algoritmo evolucionário padrão, com inserção de *random immigrants* e com inserção de *elitist immigrants*), guardando os valores correspondentes ao melhor indivíduo (em cada geração) e em cada execução. Nesta etapa, foi declarada uma lista aleatória, com a mesma dimensão do número de *runs*, de *seeds* estáticas, para a geração da população inicial, de maneira que os 3 algoritmos tenham o mesmo ponto de partida, diminuindo o impacto da influência da população caso tivesse sido criada usando números aleatórios. O facto da lista de *seeds* ser gerada aleatoriamente pode fazer com que haja a possibilidade de ocorrerem discrepâncias relativamente aos valores usados nesta análise, caso os *scripts* sejam executados novamente, pelo que os testes estatísticos devem também ser feitos de novo.

5 Resultados

Posteriormente à realização de todos os testes, foram reunidos os dados relativos às duas experiências elaboradas: testagem de melhores parâmetros para cada *benchmark* e

comparação dos 3 algoritmos evolucionários com métodos de inserção de imigrantes distintos. Com os resultados obtidos, foram efetuados testes estatísticos para avaliar se existem diferenças significativas no desempenho das diferentes versões do algoritmo. Todos os dados e imagens gerados podem ser encontrados na diretoria *output*.

5.1 Testagem de Parâmetros

De uma forma geral, para os problemas apresentados, os algoritmos que obtiveram melhores resultados possuem probabilidade de mutação baixa e, pelo contrário, probabilidade de cruzamento elevada. Constatou-se que para todos as *benchmarks*, dos valores testados, o valor mais adequado para probabilidade de mutação é 0.05 e para probabilidade de *crossover* é 0.9. Os restantes parâmetros permaneceram iguais aos presentes na *Tabela 3*, já apresentada. De seguida, na *Figura 2*, são exibidos os melhores desempenhos encontrados para cada um dos problemas. Cada gráfico representa o último valor da *fitness* em cada execução e a média dos valores médios de fitness de cada geração; quanto à *Quartic Function* é também apresentado o melhor valor de fitness encontrado em cada execução, visto que não corresponde ao último.

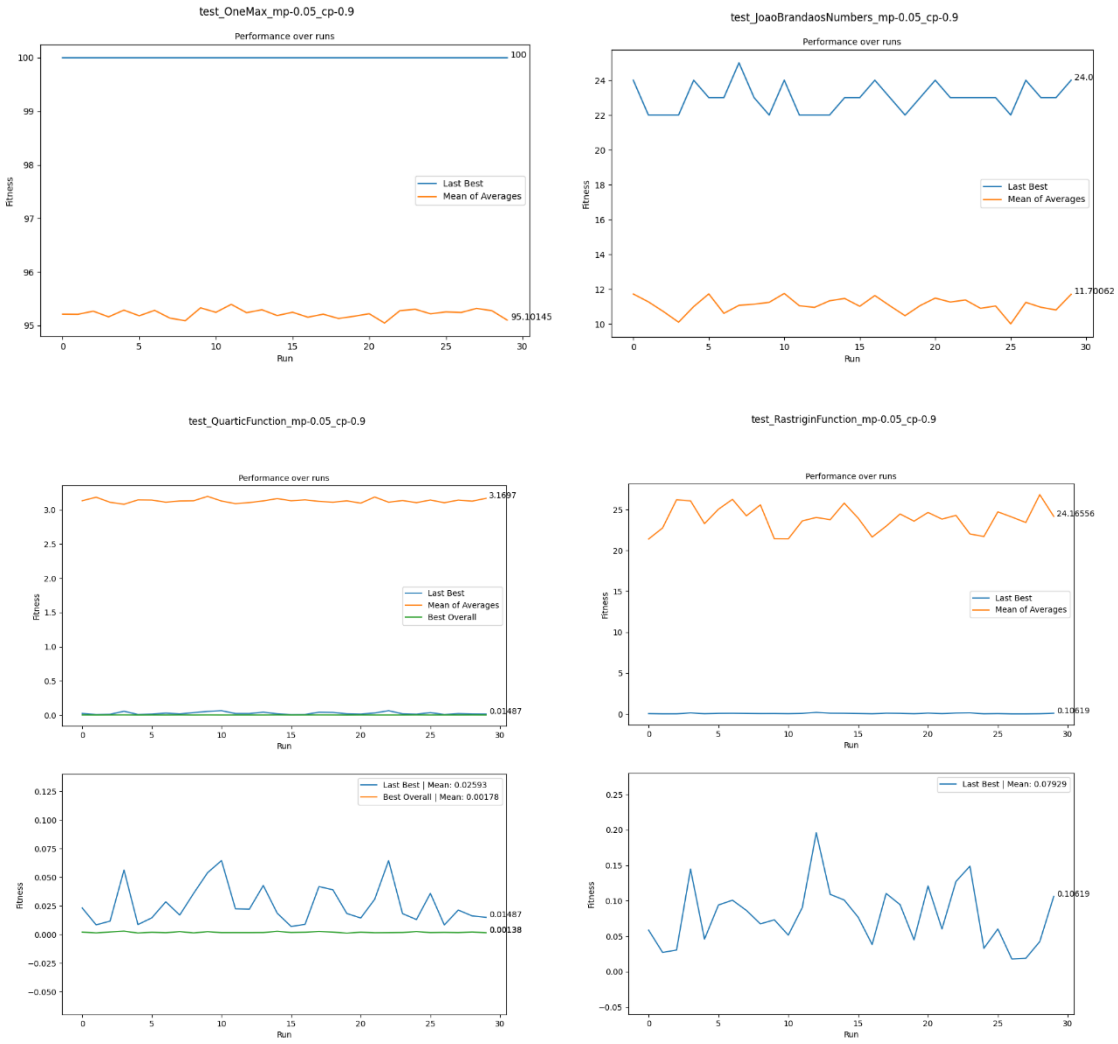

Figura 2 - Gráficos dos melhores desempenhos obtidos em cada execução, para cada problema

Após análise tanto dos gráficos acima como dos gráficos correspondentes a execuções individuais onde é possível observar a evolução das gerações, verifica-se que as *benchmarks* de menor complexidade selecionadas para cada uma das representações, *One Max* e *Quartic Function*, são solucionadas facilmente por parte do algoritmo evolutivo, com os melhores parâmetros descobertos, pelo que podem não contribuir ativamente extração de conclusões.

Para reproduzir esta experiência, pode ser executado o *script*:

parameter_testing.py

5.2 Testagem de Inserção de Imigrantes

Como esperado, todos os algoritmos evolucionários testados conseguem obter uma solução para os problemas bastante próxima da ótima. Ao comparar as 3 versões em cada geração, não se consegue identificar claras distinções entre os algoritmos, uma vez que o melhor em cada execução varia bastante. As *Figuras 3 e 4* ilustram algumas *runs* aleatórias onde é possível observar a evolução do desempenho dos algoritmos ao longo das gerações. Consegue-se inferir que, para os problemas mais simples, os valores estabilizam antes das 200 gerações; já para os mais complexos, a estagnação acontece por volta das 400 gerações, podendo, no entanto, ter pequenas melhorias até ao final, principalmente no caso de *Rastrigin Function*.

Para reproduzir esta experiência, pode ser executado o *script*:

immigrant_insertion_testing.py

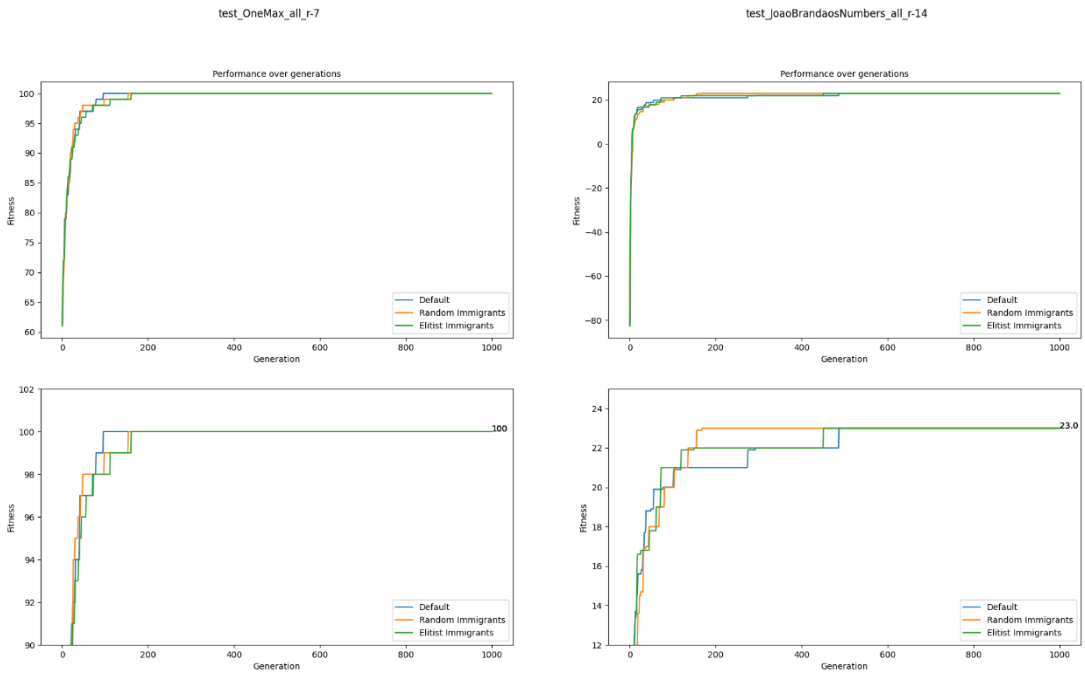

Figura 3 - Exemplo de execuções comparando os 3 algoritmos, para os problemas de representação binária

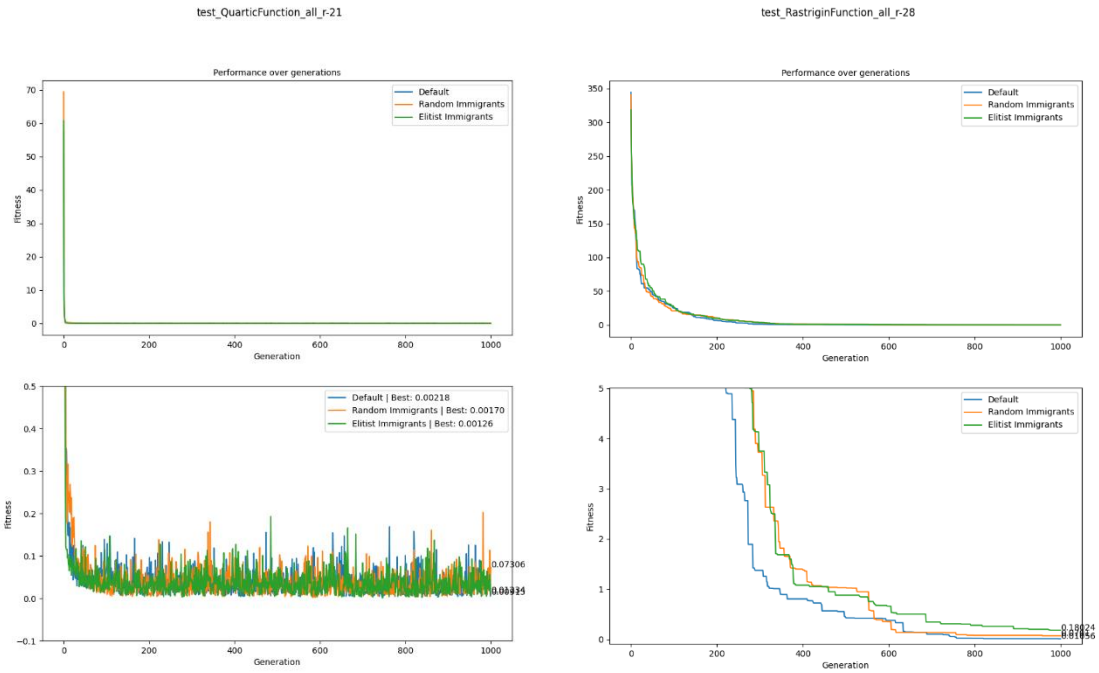

Figura 4 - Exemplo de execuções comparando os 3 algoritmos, para os problemas de representação real

6 Análise Estatística

Com os dados obtidos na última experiência, relativos ao desempenho dos algoritmos evolucionários ao longo das 30 execuções, para cada problema, foi possível retirar conclusões relativamente ao uso de técnicas de inserção de imigrantes comparativamente ao algoritmo padrão. Deste modo, foi em primeiro lugar confirmado se os dados seguem (ou não) uma distribuição normal, com recurso a *KS Test*, e aplicado posteriormente o teste mais adequado para verificar se cada um dos algoritmos com inserção de imigrantes é igual ao padrão ou se existem diferenças entre eles, usando *Wilcoxon Test*. O nível de significância usado foi definido como como α=0.05.

Tendo isto em conta, as hipóteses são as seguintes:

- *KS Test*:
  - H0: Os dados seguem uma distribuição normal.
  - H1: Os dados não seguem uma distribuição normal.
- *Wilcoxon Test*:
  - H0: Os algoritmos são iguais – não há diferenças significativas.
  - H1: Os algoritmos não são iguais – há diferenças significativas.

Os resultados dos testes estatísticos são representados abaixo usando uma tabela onde as colunas “KS Test” e “Wilcoxon” indicam os respetivos *p-values*, a coluna “Effect Size” evidencia o grau de diferença existente e a coluna “Result” indica se a hipótese nula é aceite ou rejeitada.

Para reproduzir esta análise, pode ser executado o *script*:

statistical_analysis.py

6.1 *One Max*

Uma vez que este problema é facilmente resolvido pelos 3 algoritmos em poucas gerações, todas as execuções devolvem a melhor solução. Por esse motivo, pode-se constatar que, neste caso, não existem diferenças entre o uso de inserção de imigrantes e a não utilização dessa técnica.

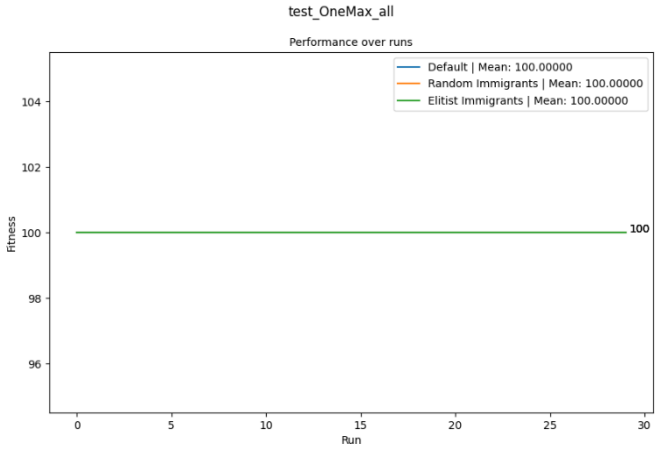

Figura 5 - Comparação dos melhores desempenhos encontrados em cada execução, para os 3 algoritmos, no problema One Max

6.2 *João Brandão's Numbers*

Neste problema, consegue-se observar que os melhores valores de fitness são atingidos por todos os algoritmos, podendo-se destacar o algoritmo padrão e com inserção de emigrantes elitistas, que atingem a melhor solução. No entanto, os valores médios nas 30 execuções, são bastante similares entre todos.

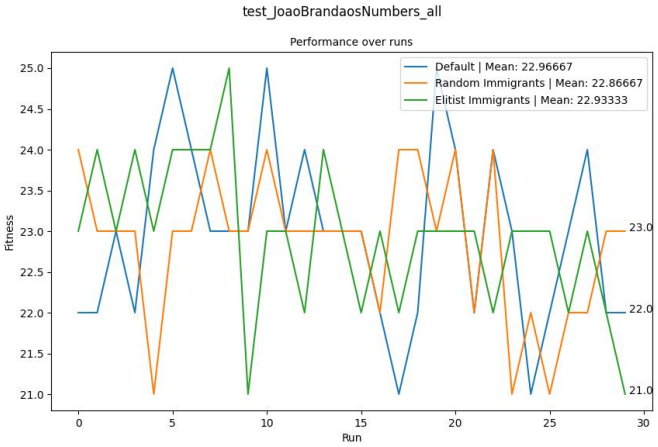

Figura 6 - Comparação dos melhores desempenhos encontrados em cada execução, para os 3 algoritmos, no problema João Brandão's Numbers

Segundo os resultados dos testes estatísticos, nenhum dos algoritmos com inserção de imigrantes possui diferenças significativas relativamente ao padrão.

Algorithm	KS Test	Wilcoxon Test	Effect Size	Result
Default	4.005e-04	-	-	-
Random Immigrants	2.363e-08	0.6346	0.39565	Accept
Elitist Immigrants	1.140e-04	0.85895	-0.235	Accept

Tabela 4 - Dados relativos à análise estatística para o problema Joaão Brandão's Numbers

6.3 *Quartic Function*

Para esta *benchmark*, o algoritmo com inserção de imigrantes elitistas é o que atinge o melhor valor de desempenho; por outro lado, o que possui melhor valor médio é aquele que recorre à inserção de imigrantes aleatórios. Apesar disso, as diferenças não são suficientemente expressivas.

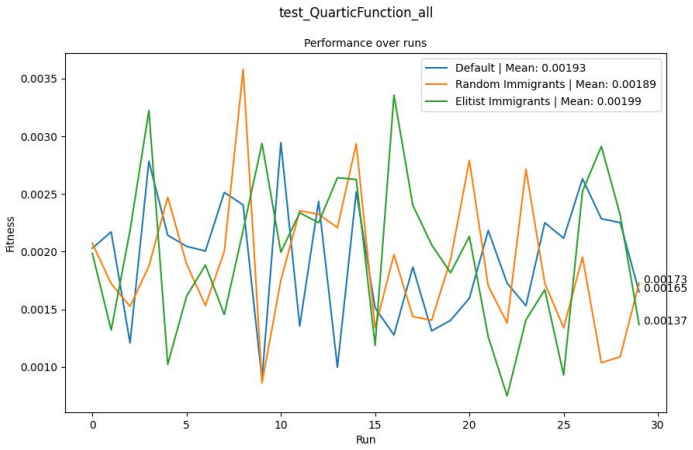

Figura 7 - Comparação dos melhores desempenhos encontrados em cada execução, para os 3 algoritmos, no problema Quartic Function

Segundo os resultados dos testes estatísticos, nenhum dos algoritmos com inserção de imigrantes possui diferenças significativas relativamente ao padrão.

Algorithm	KS Test	Wilcoxon Test	Effect Size	Result
Default	1.020e-05	-	-	-
Random Immigrants	7.822e-05	0.59781	-0.07037	Accept
Elitist Immigrants	2.494e-04	0.90323	-0.01726	Accept

Tabela 5 - Dados relativos à análise estatística para o problema Quartic Function

6.4 *Rastrigin Function*

Novamente, os algoritmos padrão e com inserção de imigrantes elitistas são aqueles que parecem atingir melhores valores de *fitness* neste problema, no entanto, os valores médios de desempenho dos 3 métodos não apresentam diferenças consideráveis.

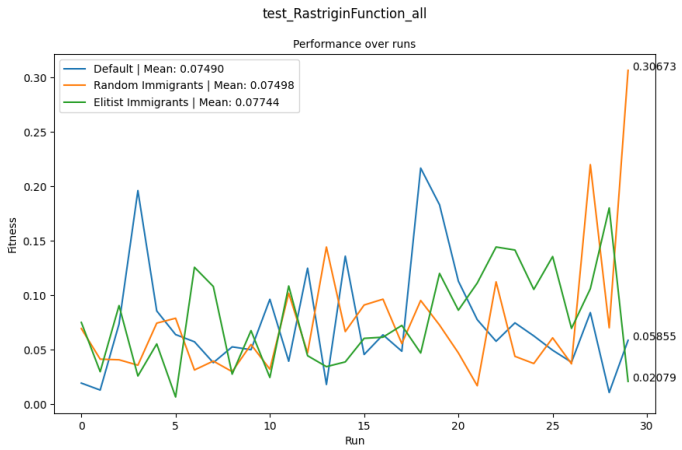

Figura 8 - Comparação dos melhores desempenhos encontrados em cada execução, para os 3 algoritmos, no problema Rastrigin Function

Segundo os resultados dos testes estatísticos, nenhum dos algoritmos com inserção de imigrantes possui diferenças significativas relativamente ao padrão.

Algorithm	KS Test	Wilcoxon Test	Effect Size	Result
Default	5.790e-07	-	-	-
Random Immigrants	1.812e-06	0.74565	-0.04381	Accept
Elitist Immigrants	4.135e-05	0.70003	-0.05178	Accept

Tabela 6 - Dados relativos à análise estatística para o problema Rastrigin Function

7 Conclusão

Com este trabalho, foi possível concluir que as técnicas de inserção de imigrantes não acrescentam diferenças significativas na *performance* do algoritmo evolutivo. No entanto, é importante reforçar que estes resultados apenas se aplicam aos dados e *benchmarks* utilizados, não podendo ser generalizáveis para outras situações. Assim, uma possível explicação para este acontecimento poderá ser que os métodos empregados não melhoram a resolução das *benchmarks* selecionadas, pelo que com o uso de outros problemas, os dados poderão ser mais esclarecedores. É relevante referir que, como aconteceu durante a realização deste estudo, a utilização de populações iniciais diferentes no algoritmo evolutivo, derivado das *seeds*, pode levar a resultados diferentes; mesmo assim, as alterações nas conclusões não devem ser muito acentuadas.

Relativamente às experiências, existem alguns aspetos que poderiam ter sido melhorados, nomeadamente a testagem usando mais parametrização e métodos diferentes para cada um dos processos referentes aos operadores de variação e de seleção de indivíduos. Além disso, nos testes dos respetivos problemas, o tamanho dos indivíduos e as dimensões poderiam variar, de maneira a introduzir um maior grau de dificuldade nos problemas, fazendo com que a inserção de imigrantes exercesse um impacto mais significativo. Outra possibilidade ainda, seria aumentar a percentagem de população a ser substituída por imigrantes. Por conseguinte, alguns dos resultados poderiam ser mais satisfatórios ou levar a outras conclusões.