

Projeto 2 - MC906A

Marcelo Biagi Martins - 183303
Erick Seiji Furukawa - 170600

1 Resumo

Algoritmos genéticos são famosos por serem formas de busca extremamente eficientes e que geram resultados incrivelmente satisfatórios, visto que a dificuldade de sua implementação é relativamente baixa em comparação com outras formas também eficientes de resolução dos mesmos problemas. Como esse paradigma de computação trabalha com evolução e implementa conceitos biológicos da vida real no computador, seu entendimento é simples e sua execução é poderosa, por isso, é um tipo de resolução de problemas intensamente estudado.

O projeto 2 de MC906A apresentado nesse relatório envolveu a resolução do problema de aproximar imagens utilizando algoritmos genéticos. Para isso, foi desenvolvido um programa que implementa as ideias de programação genética e as utiliza para aproximar uma imagem inicialmente aleatória de uma imagem alvo.

Alguns resultados importantes foram a linearidade do algoritmo desenvolvido, que apresentou o dobro do tempo de execução para uma imagem duplicada. Além disso, foi possível analisar os efeitos do tamanho da população e da taxa de mutação nos resultados. Sendo que populações/taxas de mutação muito grandes ou muito pequenas muitas vezes não convergem. Além disso, a aleatoriedade na mudança dos pixels se mostrou muito eficiente no primeiro milhão de gerações, algo que muda após essa marca.

2 Introdução

Os algoritmos genéticos são uma importante ferramenta de busca da ciencia da computacao. Inspirados pelo processo natural da Biologia, esses códigos buscam simular no computador o processo que acontece na vida real, entre os seres vivos, de seleção natural. Assim, são métodos que buscam sempre os indivíduos com as melhores características e que mais se assemelham com a função objetivo, que na vida real é a sobrevivência.

Similarmente aos organismos vivos, os algoritmos genéticos simulam reproduções entre sua população, permitindo crossover e mutação para que ocorra a “troca de genes” e para que os indivíduos que tenham os melhores resultados e cheguem mais perto da função objetivo possam progredir.

Sendo assim, escolheu-se o problema de aproximação de imagens como alvo deste projeto utilizando a linguagem de programação python. Esse problema consiste em obter uma imagem o mais próximo possível de uma outra original, utilizando um algoritmo genético. Para isso, usou-se imagens no formato RGBA e comparações constantes entre a imagem alvo e a imagem atual e era possível variar diversos parâmetros, como o intervalo entre cada coordenada RGB, o A(alfa), o tamanho da população, a taxa de mutação, a imagem inicial, entre outros. O código será explicado em maiores detalhes no decorrer do relatório.

3 O modelo

Quanto ao código, os parâmetros eram recebidos e a compilação era feita pelo comando “python3 projeto2.py imagem.formato tamanho_populacao taxa_mutacao”. As imagens utilizadas eram pequenas e fundos transparentes foram aproximados por cores aleatórias e, por isso, alfa foi fixado em 255, pois não interferia nos resultados.

Com o código compilado, seu funcionamento era simples: a taxa de mutação era responsável por mudar apenas alguns pixels e, sendo assim, apenas estes eram levados em consideração na nova função fitness, representada pela variável “score”. A função score, por sua vez, calcula a diferença entre cada pixel mutante e a imagem original e eleva esse resultado ao quadrado, pois pequenas diferenças de cor não são perceptíveis a olho nu e caso esse número não fosse elevado ao quadrado, pixels quase perfeitos poderiam ser alterados aleatoriamente por outros piores, devido a diferenças pequenas entre os resultados. Essa operação era realizada para cada canal R, G e B

de cada pixel mutante e esses resultados eram somados para criar a “score” final do pixel. Sendo assim, a função fitness busca maximizar o resultado de $1/\text{score}$. No entanto, como esse resultado pode ser muito pequeno, o código apenas tenta minimizar a score, acarretando nos mesmos resultados.

Por fim, com o score dos pixels modificados já calculados e a nova população gerada, os “tamanho_população” com o menor score do conjunto formado pela nova população e a antiga são selecionados para continuar. Dessa forma, o código continua executando por diversas gerações, até a geração 5.000.000, se necessário.



Figura 1: Imagens utilizadas nos testes. Da esquerda para a direita: um emoticon, Neptune (personagem do jogo Hyperdimension Neptunia) e uma personagem de um anime.

4 Resultados e Discussão

A função score utilizada no projeto para calcular o quão próximo uma imagem está da original é definida, como já citado, como a soma dos quadrados das diferenças entre os três canais RGB da imagem original com a imagem que está sendo evoluída. O cálculo da score, por se tratar de uma operação pixel a pixel, é um problema que exige muito processamento. Existe, no entanto, uma porcentagem muito grande de pixels que não são mutados de uma geração para outra, sendo desnecessário o processo de calcular suas scores novamente. Com esta ideia, o algoritmo desenvolvido no projeto atualiza as scores de cada indivíduo no momento da mutação, calculando o valor da nova score utilizando o valor da score antiga, e os score dos pixels mutados nesta geração. Desta forma, o cálculo de scores torna-se uma tarefa centenas, ou milhares de vezes mais rápida.

Para realizar os cálculos, valores padrão foram definidos com as imagens apresentadas. O primeiro valor de comparação utiliza a imagem “emoticon” de dimensões 240px por 240px, com população de 5 e taxa de mutação de 0,01%, começando com o fundo tendo todos os pixels pretos(valor 0). Seus dados de tempo e média de score, até a geração 1.000.000, estão na tabela 1, seu resultado final, na geração 4.000.000 está disposto na figura 2, juntamente com um gráfico que representa seu score ao longo do tempo.

Tabela 1: Execução com imagem inicial preta, população 5 e mutação de 0,01%.

Geração	Tempo	Score médio
1	2,654	7111463618
100	2,864	7097891275
10.000	12,631	5872889609
30.000	31,152	4158965502
50.000	50,708	3110441722
80.000	78,885	2214646595
110.000	106,401	1740898151
150.000	145,664	1388121168
200.000	199,961	1140584624
250.000	199,961	989518575
300.000	244,389	885958659
400.000	288,054	751929935

Geração	Tempo	Score médio
500.000	492,807	664437140
600.000	584,457	602987188
700.000	673,479	557546731
800.000	763,371	521334411
900.000	856,973	492140312
1.000.000	948,421	467071620

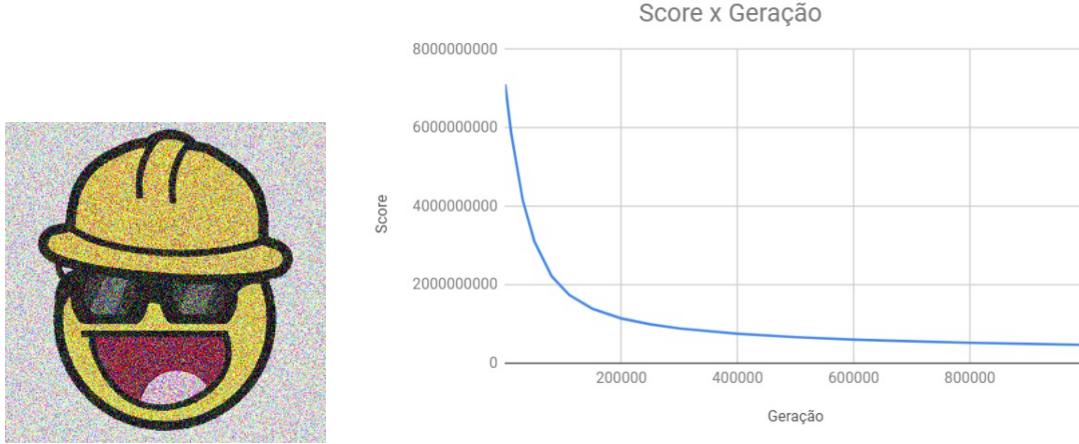


Figura 2: Imagem final da execução com fundo inicial preto e gráfico de Score em função da geração.

Assim, diversos parâmetros podem ser trocados para testar a eficiência do código e vários testes foram feitos para isso. O primeiro deles buscava analisar a diferença entre a forma com que a imagem inicial era gerada. Para isso, comparou-se a diferença no tempo de execução no caso da imagem inicial ser inteira preta e no caso em que ela é inteira aleatória. Os resultados obtidos estão representados na figura 3. Dessa forma, a inicialização constante se mostrou razoavelmente mais eficiente do que a aleatória. Apesar de ser uma diferença sutil, a inicialização constante com cor preta foi a escolhida para os outros testes. Além disso, os resultados finais também não apresentaram grandes diferenças e estão dispostos na figura 4, demonstrando que a diferença é praticamente imperceptível a olho nu.



Figura 3: Gráfico de tempo em função de geração para inicialização aleatória e constante.



Figura 4: Comparação entre imagem da geração 1.000.000 para fundo preto e aleatório.

O segundo teste, por sua vez, verificava se a duplicação das dimensões da imagem acarretaria em um tempo quatro vezes maior de execução, como esperado. Assim, a imagem "emoticon" foi duplicada e passou a ter dimensões de 480px por 480px. A tabela 2 mostra o tempo de execução para a imagem original e a imagem duplicada e o gráfico 5 descreve a tabela.

Tabela 2: Execução com imagem inicial preta, população 5 e mutação de 0,01% para imagem normal e duplicada

Geração	Tempo imagem emoticon(s)	Tempo imagem duplicada(s)	Razão
1	3,681	16,385	4,45
100	3,815	17,061	4,47
10.000	14,171	66,981	4,72
30.000	31,564	157,421	4,98
50.000	48,008	245,856	5,12
80.000	78,007	390,501	5,00
110.000	105,073	536,931	5,11
150.000	145,688	718,104	4,92
200.000	197,814	946,601	4,78

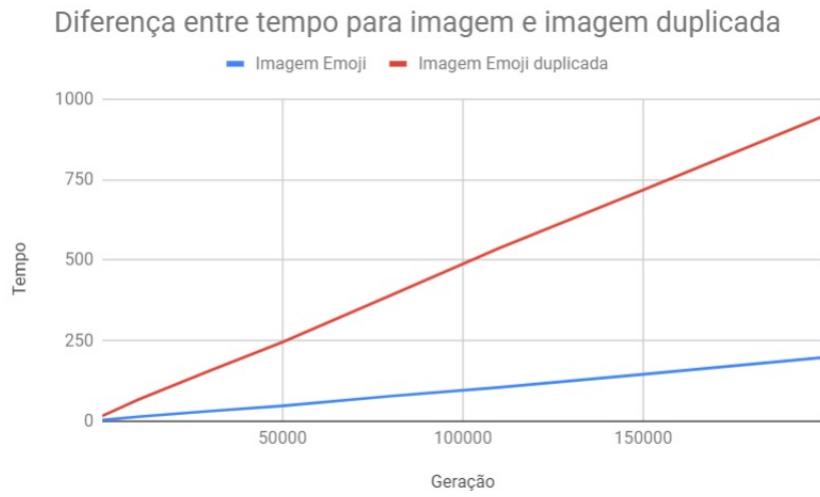


Figura 5: Comparação entre tempo de imagem emoticon e imagem emoticon duplicada.

A média das razões foi de 4,84, ou seja, mais do que o valor esperado de quatro. Porém, como o tempo de execução varia razoavelmente, devido as aleatoriedades envolvidas, e aproximações decimais são feitas, pode-se dizer que o tempo de execução do código é linear no tamanho da imagem.

O terceiro teste realizado, por sua vez, variava a taxa de mutação do programa. Como a taxa padrão foi definida como 0,01%, os resultados dessa execução foram comparados com resultados que mantinham a imagem emoticon, população de 5, mas variavam a taxa de mutação, sendo que um teste possuía taxa de 0,1% e o outro de 0,002%. Os resultados obtidos estão representados nos gráficos 6 e nas imagens 7 e 8 a seguir:

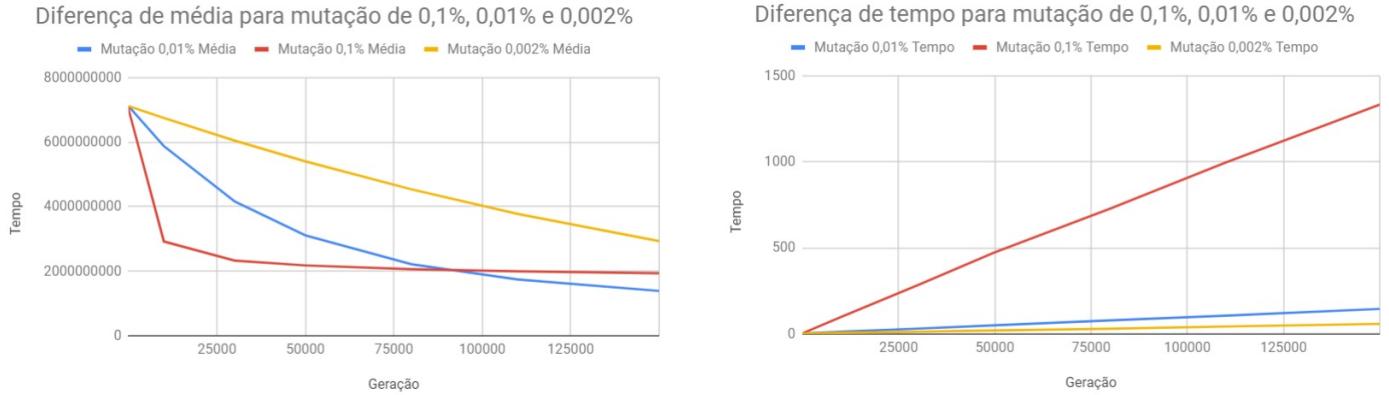


Figura 6: Diferença de tempo e média para diferentes taxas de mutação, até a geração 150.000

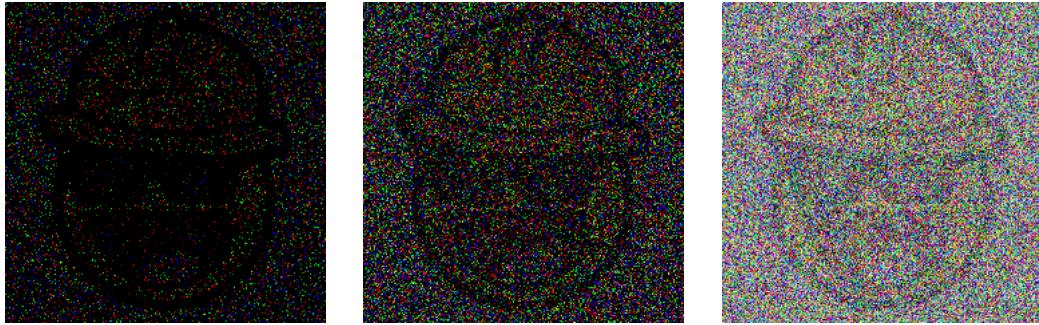


Figura 7: Imagens geradas na geração 10.000 para taxas de 0,002%, 0,01% e 0,1%, respectivamente

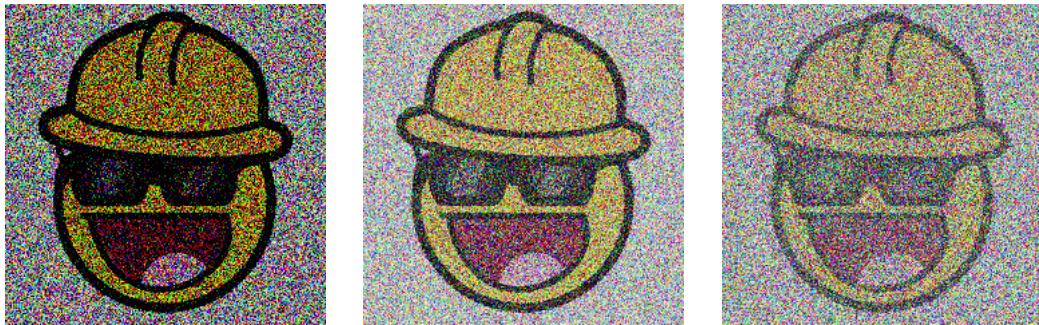


Figura 8: Imagens geradas na geração 150.000 para taxas de 0,002%, 0,01% e 0,1%, respectivamente

Assim, pode-se concluir que uma taxa de mutação muito baixa (no caso, 5) apresenta resultados satisfatórios, mas que podem demorar muito para convergir devido à baixa quantidade de pixels que mudam por iteração. Além disso, taxas de variação muito altas com populações ainda pequenas não são satisfatórias a longo prazo, pois como os pixels mudam de forma aleatória, a chance dessa mudança ser mais próxima da imagem objetivo do que a imagem anterior é muito baixa. Assim, a taxa intermediária de 0,01% foi a que apresentou os melhores resultados, tanto em tempo, quanto em média a partir da geração 90.000.

O quarto teste, por sua vez, variava a população, mantendo os outros valores, a imagem emoticon e a taxa de mutação de 0,01% fixos para poder verificar a influência da população na eficiência do código. Os resultados obtidos estão nos gráficos 9 e nas imagens 10 e 11 a seguir:

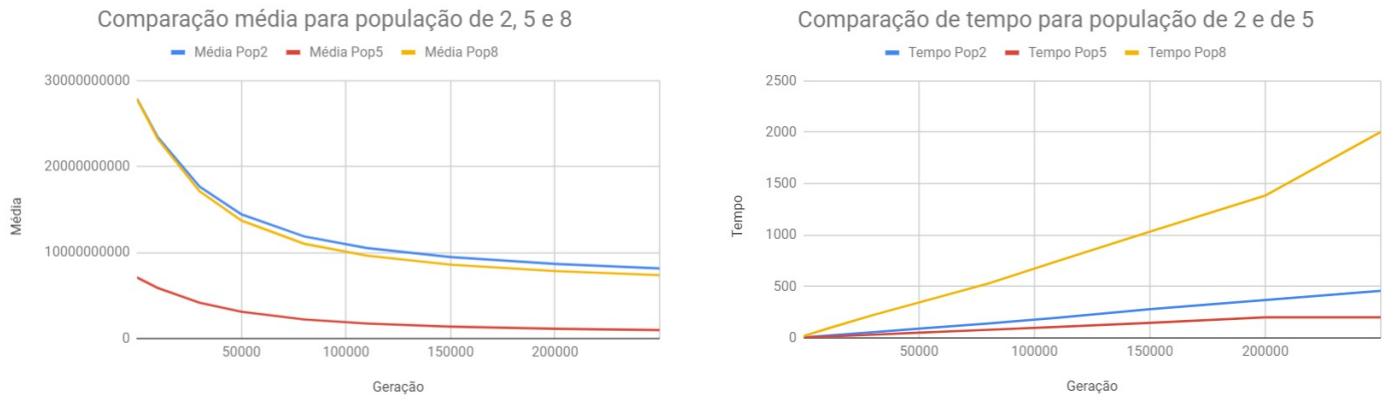


Figura 9: Comparação de tempo e média para diferentes taxas de mutação, até a geração 150.000

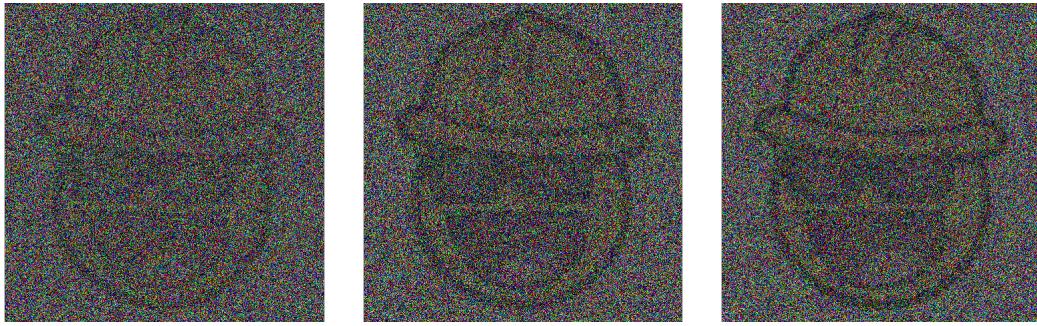


Figura 10: Imagens geradas na geração 30.000 para populações de 2,5 e 8, respectivamente

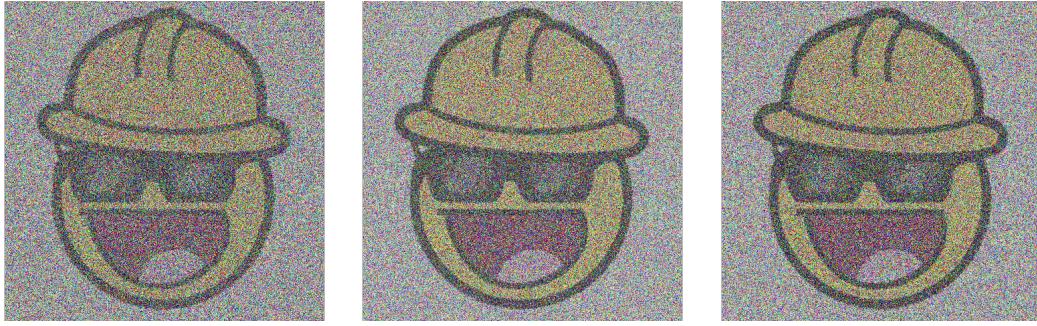


Figura 11: Imagens geradas na geração 200.000 para populações de 2,5 e 8, respectivamente

A partir dos dados obtidos, podemos concluir que populações muito pequenas não permitem a variação necessária e, por isso, os resultados demoram muitas gerações para se aproximar da imagem objetivo.

Teoricamente, populações maiores devem fazer com que o problema venha a convergir em um número de gerações menor, que pode ser uma vantagem em alguns casos. O problema está no fato de que o aumento da população, gera um aumento muito grande no tempo necessário para o processamento do problema. Como pode ser visto no gráfico de tempo da figura 9, o tempo necessário para executar o problema com uma população de oito, chega a ser quatro vezes maior que o problema com uma população cinco. Além disso, para este caso específico de aproximação de imagens, o resultado obtido não variou significativamente.

Para a realização do quinto teste, foram utilizadas diferentes formas de mutação e substituição:

- Aleatória

A mutação aleatória determina valores aleatórios no intervalo de (0,255) para os canais RGB de um pixel.

- Quantizada

A mutação quantizada também determina o valor da intensidade de um pixel aleatoriamente, porém um pixel só pode assumir um valor múltiplo de 8. Dessa forma, o número de cores possíveis é reduzido para 32, de 256 cores. A ideia deste método de mutação é que o olho humano não consegue distinguir diferenças na imagem se o valor dos pixels diferirem por um valor muito pequeno (no caso deste método de mutação, até sete unidades de diferença).

- Incremental

A mutação incremental altera o valor dos pixels somando ou subtraindo um número aleatório no intervalo de (1,10) no valor de um pixel. Isso faz com que os pixels se alterem aos poucos com o passar das gerações, e não com mudanças totalmente aleatórias.

- Híbrida

A mutação híbrida é a junção das mutações quantizada e da mutação incremental. A imagem é mutada até a geração um milhão com a mutação quantizada, e a partir daí muta-se com a mutação incremental.

Na figura 13 pode-se observar o resultado dos diferentes métodos de mutação na imagem da Neptune.

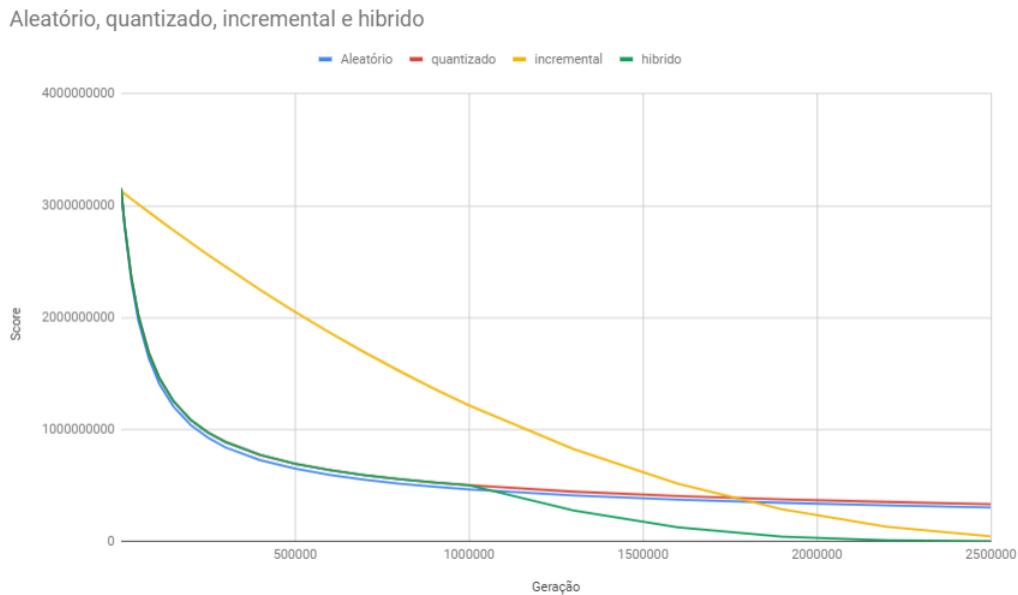


Figura 12: Gráfico da score por geração dos diferentes métodos de mutação.



Figura 13: Imagens geradas na geração 2.500.000 utilizando os métodos de mutação aleatória, quantizada, incremental e híbrida, respectivamente.



Figura 14: Imagem da geração 2.500.000 na esquerda, e imagem da geração 5.000.000 na direita.

Os métodos de mutação aleatória e mutação quantizada produziram resultados não muito bons, e apesar da imagem da mutação quantizada possuir oito vezes menos cores possíveis para mutar, este método não produziu resultados mais rapidamente. A princípio acreditava-se que por causa do número de cores ser oito vezes menor, a imagem iria se aproximar mais rapidamente da imagem original, mesmo que os pixels possuíssem uma diferença de até sete unidades da cor ideal.

Observa-se que a mutação incremental produziu um resultado muito melhor que os dois métodos anteriores. A partir do gráfico da figura 12, nota-se que a mutação incremental demora bem mais para produzir resultados, porém ela não estagna tão rapidamente como a mutação aleatória ou quantizada, o que faz com que produza um resultado melhor quando executada por um tempo adequado.

Para tentar produzir um resultado ainda melhor foi utilizado o método de mutação híbrido, que possui as vantagens da mutação quantizada que é rápida para aproximar a imagem no início (quase tão boa quanto o método de mutação aleatória), e da mutação incremental que não estagna rapidamente. Com isso pôde-se obter uma imagem praticamente igual a original.

Por fim, foi realizado uma execução até a geração 5.000.000 para ver até onde o algoritmo conseguia aproximar a imagem da original. Para este teste, foi utilizada a imagem do personagem de anime da figura 1, e o método de mutação híbrido que mostrou ser o mais eficiente de todos.

Observa-se na figura 14 que as imagens estão praticamente perfeitas, contendo diferenças imperceptíveis da imagem original. A score da geração 2.500.000 é de 1557917, e a score da geração 5.000.000 é 465450. A execução do algoritmo levou uma hora e 51 minutos.

5 Conclusões

Através dos resultados obtidos, podemos concluir que todos os parâmetros possuem total influência nos resultados da execução. Primeiramente, o tamanho da imagem influencia no tempo de forma linear, ou seja, imagens duas vezes maior levam o dobro do tempo para serem processadas e para atingirem resultados semelhantes.

Além disso, a população e a taxa de mutação definem, em conjunto, se o código converge ou não, sendo que uma discrepância muito grande de seus valores geralmente acarreta em uma execução divergente, pois altas taxas de mutação com população pequena não se aproximam da imagem objetivo devido a aleatoriedade do sistema; e baixas taxas de mutação com população alta não promove a diversidade necessária para chegar ao resultado esperado. Além disso, esses casos contribuem para um aumento no tempo de execução.

O método de mutação também é um fator que pode gerar bastante diferença no tempo de execução do algoritmo. Se os métodos de mutação não forem bem pensados, o algoritmo pode vir a estagnar em um momento cuja solução ainda não é satisfatória.

Também foi observado que mudar a técnica de mutação a partir de certa geração gerou resultados muito mais satisfatórios do que utilizando apenas uma técnica ou outra. Levando em conta esta ideia, pode ser uma boa estratégia realizar também a alteração das taxas de mutação e do número da população a partir de uma certa geração.

Referências

- [1] *Partes de um algoritmo genético*, Disponível em https://pt.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico