Aplicação de Algoritmo Genético para o problema do Caixeiro Viajante

Allan Diamante de Souza, 105423

Felipe Diniz Tomás, 110752

[1]
Departamento de informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

Modelagem e otimização Algorítmica – 6903
Ra105423@uem.br, Ra110752@uem.br

18, Maio 2021

1. Introdução

O conceito de Algoritmo Genético nasce com Ingo Rechenberg[1], um cientista da computação Alemão, no ano de 1960, ele discorre estratégias sobre o campo da computação evolutiva no seu artigo "Estratégias de Evolução". Porém, o tema toma mais relevância com John Henry Holland [2] após se aprofundar e desenvolver algoritmos juntamente com seus colegas e alunos.

O Algoritmo Genético é uma técnica para resolução de problemas de otimização e busca, que utiliza inspirações pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação. Esse método é aplicado baseando-se em uma codificação do conjunto das soluções possíveis, retornando o melhor caso encontrado a partir de um tempo.

Sendo assim, o objetivo do trabalho é desenvolver, analisar e compreender o funcionamento do Algoritmo Genético para a resolução do problema do Caxeiro viajante, além de analisar a combinação de busca local ao mesmo. Os enfoques observados serão, tempo de execução, distancia total entre as cidades (solução), quantidade de gerações, quantidade de população, método de seleção, taxa de mutação, e por fim destacar a implementação da busca local e técnica de geração de vizinhos.

2. O problema

O problema do Caxeiro Viajante [3] é um dos mais famosos problemas combinatórios que existe, consiste em um viajante visitar n cidades diferentes, iniciando e encerrando sua viagem na cidade de partida inicial. A total liberdade na ordem de visita e também se supõem que haja ligações diretas entre todas as cidades, porém, o viajante deve passar apenas uma vez em cada cidade. O retorno do algoritmo para resolver o problema do caxeiro consiste em descobrir a menor rota existente, sendo esse valor a soma do peso de cada caminho que ele percorreu.

---

[1] https://en.wikipedia.org/wiki/Ingo_Rechenberg
[2] https://pt.wikipedia.org/wiki/John_Henry_Holland
[3] https://en.wikipedia.org/wiki/Travelling_salesman_problem

Cosidera-se que o problema é NP-difícil, sendo formulado pela primeira vez em 1930, destaca-se como um dos problemas de otimização mais intensamente estudados.

3. Algoritmo Genético

Algoritmos Genéticos (AG) são implementados de forma em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução da população geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente e é realizada por meio de gerações.

O algoritmo segue uma estrutura de etapas baseada na teoria evolutiva, como pode se oberservar na Figura 1.

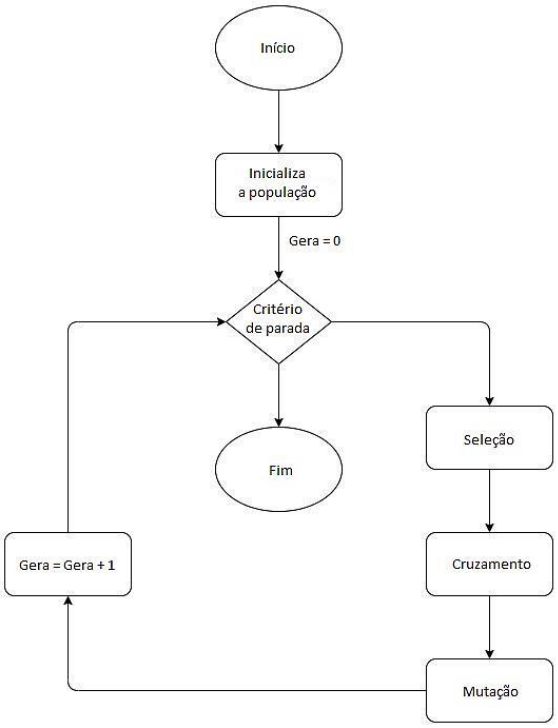

Figura 1. Diagrama do Algoritmo genético.

Para melhor compreensão da implementação desta etapa, é necessário relembrar alguns conceitos importantes da definição do algoritmo genético, são eles:

- Indivíduo – É um portador de um código genético. Sendo o código genético uma representação do espaço de busca do problema a ser resolvido.
- Seleção – É uma parte chave do algoritmo, a seleção é reponsável por selecionar os indivíduos que passarão pelo processo de cruzamento.
- População – Conjunto de indivíduos.
- Geração – Uma iteração completa do AG sendo o seu retorno uma nova população.
- Elitismo - Consiste em reintroduzir o indivíduo melhor avaliado de uma geração anterior para a seguinte, evitando a perda de informações importantes presentes em indivíduos de alta avaliação e que podem ser perdidas durante os processos de seleção e cruzamento, caso não o houvesse.
- Cruzamento ou *Crossver* - Cruzamento troca arbitrariamente os genes entre dois indivíduos selecionados para reprodução.

- Mutação - Mutação é uma operação exploratória que tem por objetivo aumentar a diversidade na população.

3.1 População Inicial

A população inicial é composta por caminhos distintos aleatórios. Cada caminho único portanto, é um indivíduo. No algoritmo a representação de um indivíduo, se dá pela classe Route, que nada mais é uma rota que passa por todas as cidade (classe Node) do nosso problema.

A geração da população inicial finalizará quando batermos o limite de indivíduos, sendo assim o tamanho da população é uma variável setada dentro do código. O custo de execução da geração da população inicial é de $O(n)$, e apenas é executada uma única vez. A biblioteca *random*[4], padrão do pyhton foi aplicada para criar os caminhos aleatórios, além disso a população é representada em uma estrutura de heap minímo, utilizando a biblioteca *heapq*[5].

3.2 Função Fitness

Também chamada de função de aptidão, é uma função objetiva que dá a informação de quão bom o indivíduo é para uma seleção posterior. Em nosso problema, a aptidão passa a ser a distância total da rota, sendo que a distância entre cada cidade se dá pelo cáculo euclidiano:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

O python possui uma biblioteca padrão *math*[6] que possibilita o cálculo da distância euclidiana de forma mais otimizada, e foi aplicado ao projeto. A complexidade de tempo do cálculo da função fitness é $O(n)$.

3.3 Seleção

A seleção é uma etapa essencial para eficiência do algoritmo genético, já que serão escolhidos os indivíduos que passarão pelo processo de cruzamento na próxima etapa. Existem vários métodos para realizar a selação, o escolhido nesta implementação foi a seleção por roleta[7].

Esse método consiste em uma roleta que será divídida entre todos os indivíduos da população de acordo com seu Fitness, ou seja, os indivíduos com maiores fitness possuem maior propabilidades de serem selecionados, enquanto os de menores possuem menor propabilidade.

Porém a implementação convencional da seleção por roleta não é das mais eficientes em questão de tempo. Em nossa primeira implementação, graças a biblioteca *cProfile*[8]foi possível constatar que a seleção era o processo que mais custava tempo em execução.

Sendo assim, graças ao um artigo desenvolvido pelos pesquisadores (Adam Lipowski, Dorota Lipowska - 2011), foi encontrado um método mais eficiente que

---

[4] https://docs.python.org/3/library/random.html
[5] https://docs.python.org/3/library/heapq.html
[6] https://docs.python.org/3/library/math.html
[7] https://en.wikipedia.org/wiki/Fitness_proportionate_selection
[8] https://docs.python.org/3/library/profile.html

produz o mesmo resultado em tempo constante, conhecido como *Roulette-wheel selection via stochastic acceptance.* Isso tornou a implementação da seleção por roleta muito mais rápida, tendo tempo constante $O(1)$.

3.4 Elitismo

O elitismo consiste em reintroduzir o indivíduo(s) melhor avaliado de uma geração anterior para a seguinte, evitando a perda de informações importantes presentes em indivíduos de alta avaliação. Na implementação foi aplicado o conceito de elitismo em cada geração definindo como 25% do tamanho total da população, selecionado portando os 25% melhores.

3.5 Cruzamento

Após termos a seleção dos dois indivíduos eles irão para etapa de reprodução, essa fase é chamada de cruzamento e seu objetivo é a mistura de informação genética aos seus descendentes.

O cruzamento assegura que a nova geração seja totalmente nova, mas possui, de alguma forma, características de seus pais, ou seja, a população se diversifica e mantém características de adaptação adquiridas pelas gerações anteriores.

Quando se trata de indivíduos que são caminhos, o processo de cruzamento ocorre escolhendo aleatóriamente um subconjunto do primeiro pai e preenchendo o resto com os genes do segundo pai, sem duplicar genes.

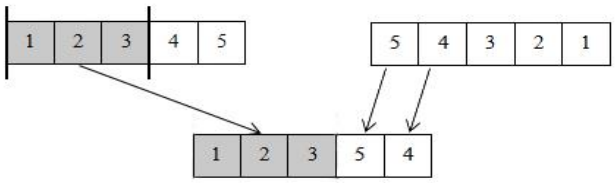

Figura 2. Exemplo de cruzamento.

O cruzamento tem tempo constante $O(1)$ já que não há complexidade alta neste processo.

3.6 Mutação

Esta fase simplesmente modifica aleatoriamente alguma característica do indivíduo sobre o qual é aplicada. Esta troca é importante, pois acaba por criar novos valores de características que não existiam ou apareciam em pequena quantidade na população em análise.

O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população. Desta forma, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca possivelmente não será zero. O operador de mutação é aplicado aos indivíduos através de uma taxa de mutação pequena, geralmente entre 0.01 a 0.3.

O processo de mutação de um caminho é feito trocando de lugar duas cidades de forma aleatória. A figura três representa um exemplo. É um processo de custo $O(n)$, por gerar núemros aleatórios.

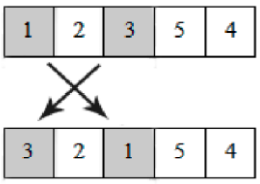

Figura 3. Exemplo de mutação.

4. Busca Local

A busca local é um método heurístico para resolver problemas de otimização computacionalmente difíceis. Pode ser usada em problemas que procuram encontrar uma solução que minimize um critério entre várias soluções candidatas. O algoritmo de busca local começa a partir de uma solução candidata e, em seguida, move-se iterativamente para uma solução vizinha. Essa geração de soluções vizinhas podem ser feita de diversas formas, utilizando heurísticas que gerem soluções.

Para a aplicação da estratégia de busca na vizinhança foi estabelecido a utilização do modelo *First Improvement*, que segundo pesquisa mostra-se melhor quando aplicado a soluções que já houveram um melhoramento[9]. De forma geral o custo da busca local será determinado pelo algoritmo de geração de vizinhos, que veremos a seguir.

4.1 Geração de vizinhos

A geração de vizinhos será o processo de busca de uma nova solução para o problema, para isso é necessário definir a qual algoritmo usar para gerar essa nova solução vizinha. Na literatura aponta-se o algoritmo 2-opt[10] como uma boa opção para o problema do caixeiro viajante, portanto foi o método selecionado para a geração de vizinhos na busca local.

Contudo o 2-opt é um algoritmo é custoso, aumenta consideravelmente o tempo da aplicação. Seu custo é de $O(n^2)$ utilizado em cada geração.

5. Metodologia

5.1 Máquina

Foi utilizado uma máquina pessoal com as configurações a seguir:

- Processador: Intel(R) Core(TM) i7-3770k CPU @ 3.50GHz;
- Memória RAM: 16.0GB;
- Disco: 100.6GB;
- GPU: RX580 8gb;

5.2 Linguagem

A linguagem utilizada foi o *python* 3.8[11] .

[9] https://www.sciencedirect.com/science/article/pii/S0166218X05003070
[10] https://en.wikipedia.org/wiki/2-opt
[11] https://docs.python.org/

5.3 Métrica de tempo

Para calcular o tempo de execução foi utilizado o módulo *time.time()*[12]

6. Resultados

Foram selecionados algumas das entradas disponíbilizadas no moodle da biblioteca TSPLIB, para testar a eficiência da implementação. Os parâmetros do algoritmo genético para os testes foram estabelecidos da seguinte forma:

- Tamanho da população: 200
- Taxa de mutação: 0.01
- Número de gerações: 300
- Tamanho da elite: 25% do tamanho da população.

A seguir a Tabela 1 contém os resultados da melhor solução entre 4 tentativas para cada caso de teste. "—" indica testes que excederam 24h.

Caso	MS	Alg	Tempo Alg	GAP_1%	Alg_bl	Tempo Alg_bl	GAP_2%
att48	10628	40903	3.848 s	284.860	34210	5.391 s	221.885
kroA100	21282	67648	7.972 s	217.864	23282	13.352 s	9.39761
a280	2579	21571	24.877 s	736.409	3029	148.93 s	17.448
rat575	6773	81925	72.348 s	1109.582	7874	1727.9 s	16.255
pr1002	259045	5204525	178.11 s	1909.12	304874	6831.40 s	17.691
tsp225	3916	24178	18.75 s	517.415	4466	83.806 s	14.044
pcb1173	56892	1170958	242.102 s	1958.212	68065	16414 s	19.638
pla33810	66048945	—	—	—	—	—	—
pla85900	142382641	—	—	—	—	—	—

Tabela 1. Resultados

6.1 Gráficos

Nesta sessão iremos ver cada gráfico gerado para cada caso de teste, tanto aplicando busca local como sem busca local. Para gerar os gráficos foi utilizado a biblioteca externa *matplotlib*[13]. No entanto, para montar o executável foi necessário comentar as chamadas da biblioteca, pois não estava aceitando a mesma. Sendo assim, para que os gráficos sejam mostrados, é necessário instalar a biblioteca e descomentar as linhas: 3, 198, 206, 207, 208, 209, 210.

[12] https://pypi.org/project/ipython-autotime/

[13] https://pypi.org/project/matplotlib/

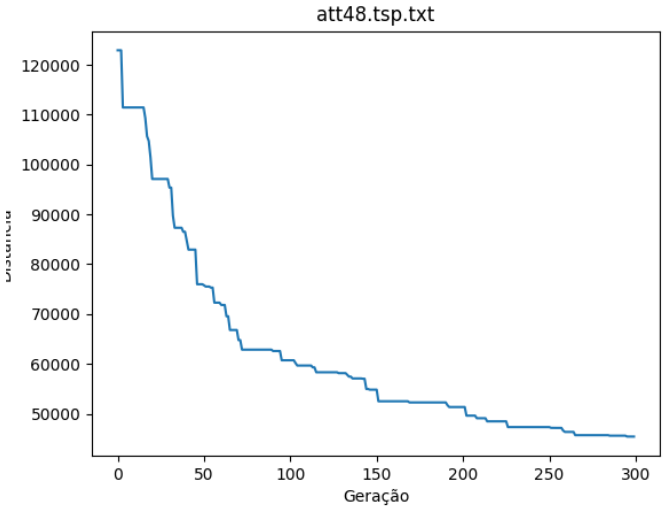

Gráfico 1. Att48.tsp (geração, distância) sem busca local.

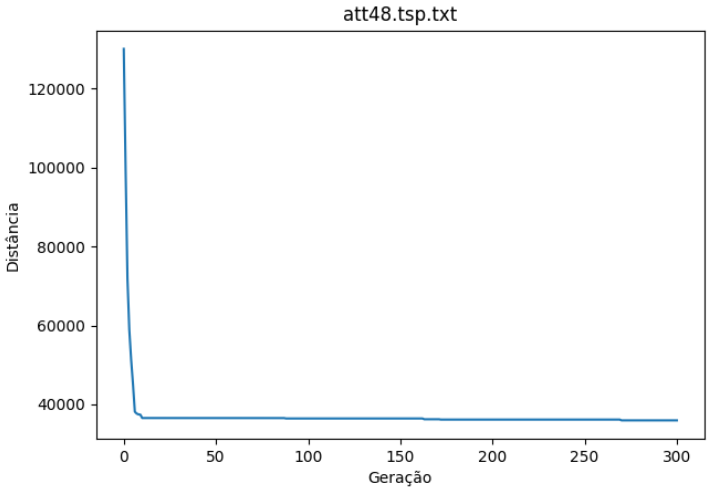

Gráfico 2. Att48.tsp (geração, distância) com busca local.

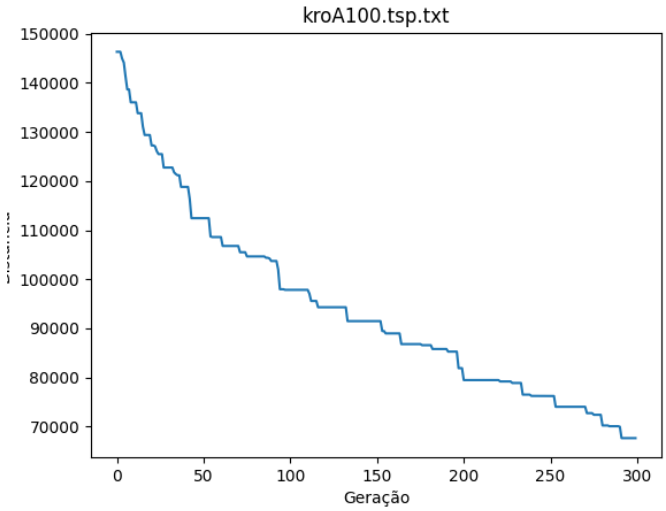

Gráfico 3. kroA100.tsp (geração, distância) sem busca local.

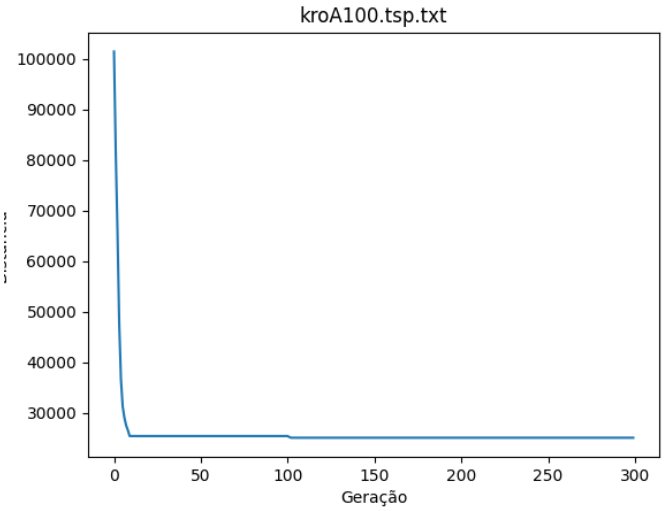

Gráfico 4. kroA100.tsp (geração, distância) sem busca local.

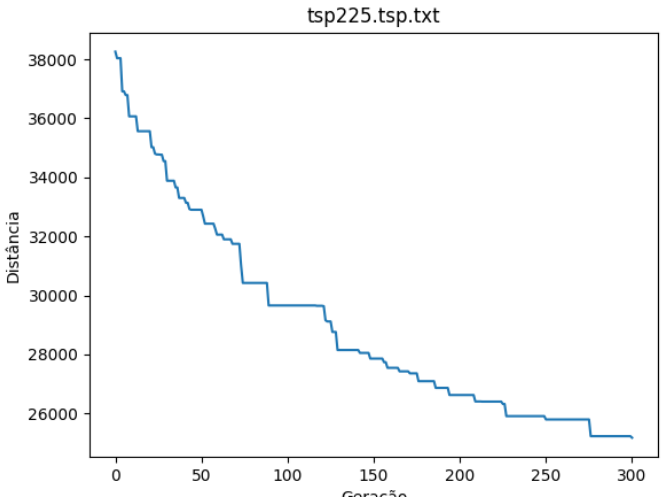

Gráfico 5. Tsp225.tsp (geração, distância) sem busca local.

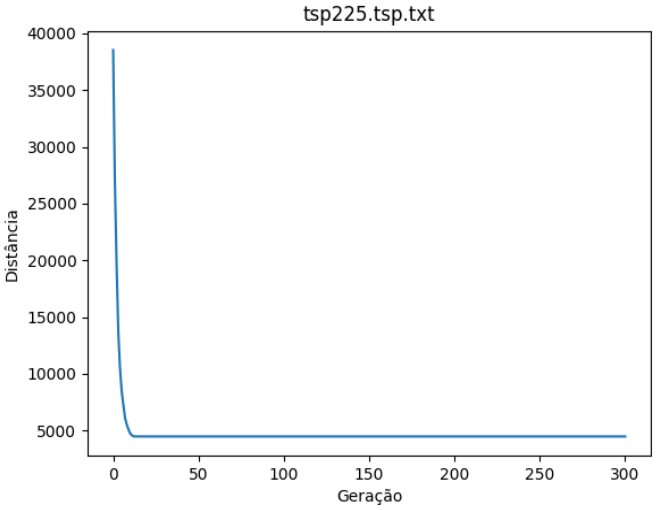

Gráfico 6. Tsp225.tsp (geração, distância) com busca local.

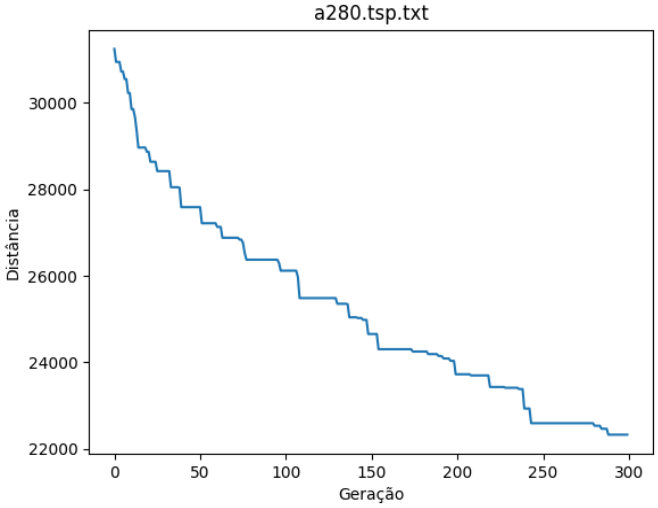

Gráfico 7. Tsp280.tsp (geração, distância) sem busca local.

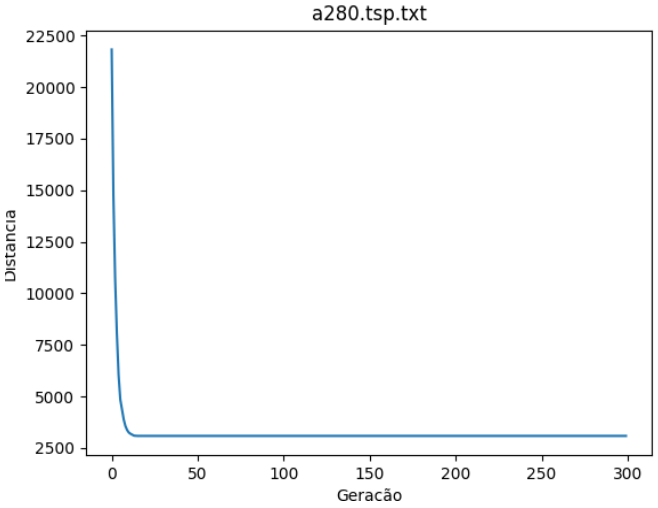

Gráfico 8. Tsp280.tsp (geração, distância) com busca local.

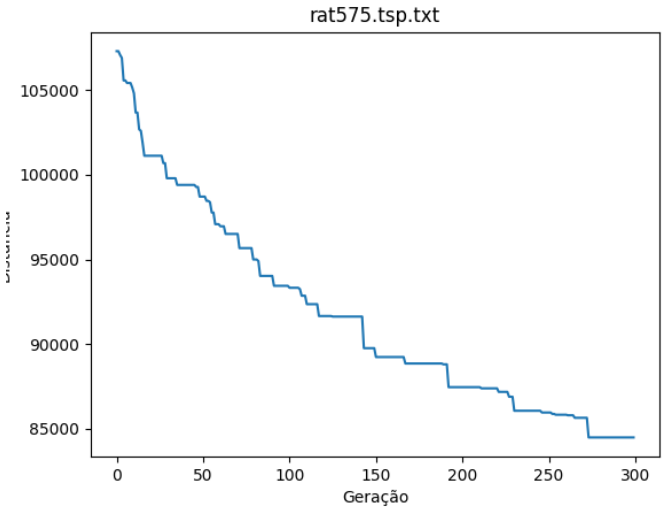

Gráfico 9. Rat575.tsp (geração, distância) sem busca local.

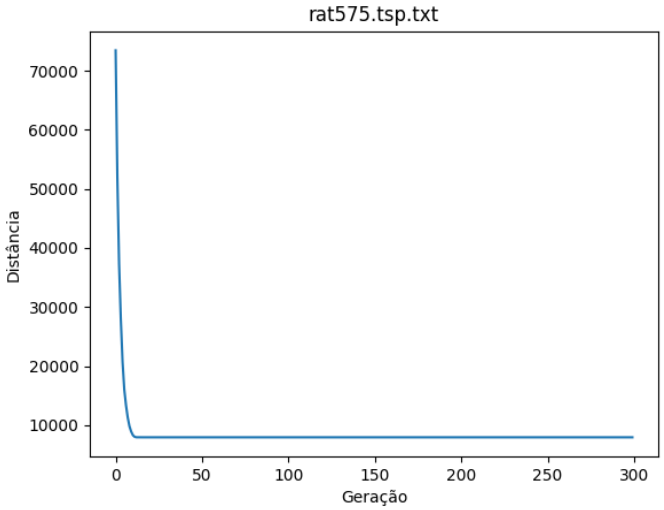

Gráfico 10. Rat575.tsp (geração, distância) com busca local

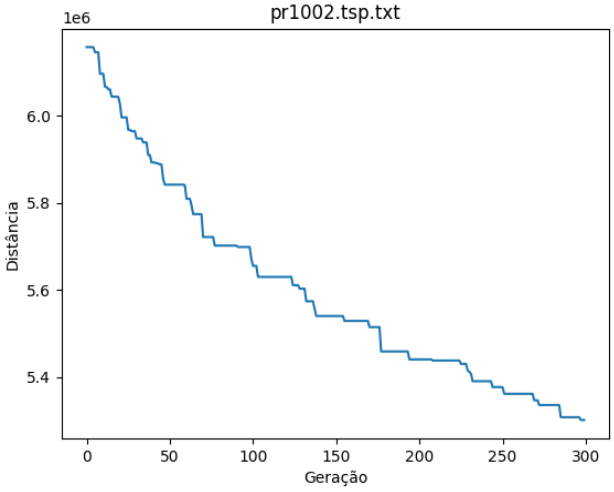

Gráfico 11. Pr1002.tsp (geração, distância) sem busca local.

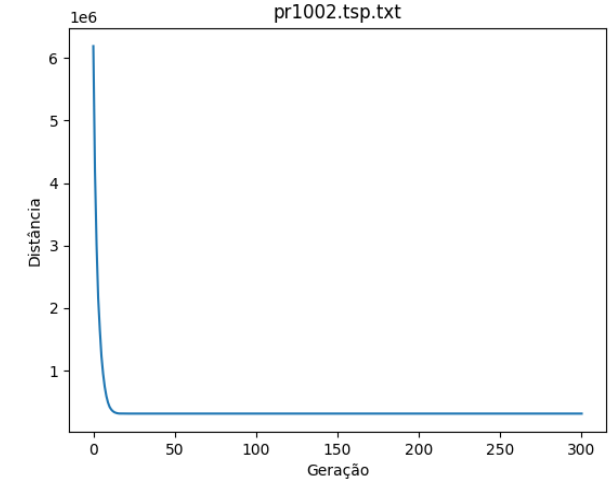

Gráfico 12. Pr1002.tsp (geração, distância) com busca local.

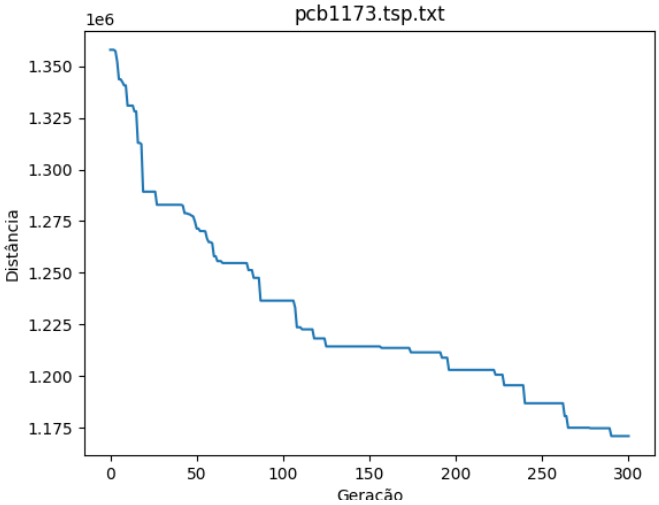

Gráfico 13. Pr1173.tsp (geração, distância) sem busca local.

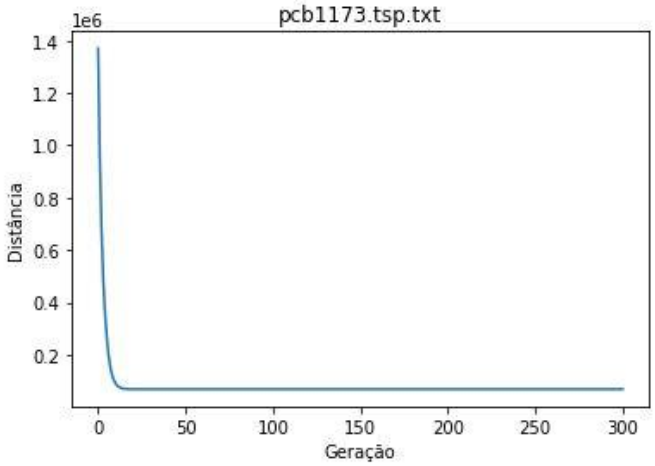

Gráfico 14. Pr1173.tsp (geração, distância) com busca local.

7. Conclusão

Percebe-se que muitos fatores influênciam nos resultados deste projeto, tanto para a solução do problema, como para o tempo de execução. Quando se trata da implementação é essencial uma pesquisa sobre as possíveis formas de estruturação de dados do problema. O heap minímo possibilita uma otimização de tempo de execução considerável aplicado ao armazenamento e manipulação dos melhores indivíduos da população.

Além disso, foi identificado que a escolha e forma de implementação do método de seleção no algoritmo genético causa uma grande influência tanto na solução como principalmente no tempo de execução. Como já comentado anteriormente, a seleção por roleta tradicional é consideravelmente mais lenta que sua versão melhorada *Roulette-wheel selection via stochastic acceptance*, o que possibilitou uma melhor eficiência quando aplicada. De maneira geral a seleção por roleta garante escolhas aletórias mais seguras em comparação com outros métodos de seleção.

Outro fator de extrema importância são os parâmetros do algoritmo genético, que definem o tamanho da população, taxa de mutação etc. Quanto maior a população, maior será o tempo de execução, no entanto a probabilidade de repetição de soluções é menor. Quanto maior a taxa de mutação, mais rápido será a execução, no entanto pior será a solução. Já se muitas gerações forem criadas, possívelmente melhor será a solução, contudo o consumo de tempo será maior. Todas essas combinações podem fazer o algoritmo variar muito suas soluções e tempo de execução.Vale ressaltar que a aplicação do elitismo a cada geração proporcionou melhores resultados.

Por fim a busca local, como já explicado, aumenta consideravelmente o tempo de execução, o que pode fazê-la inadequada para alguns casos. A opção *First Improvement* foi a que se mostrou melhor combinado ao algoritmo 2-opt.

Para projetos futuros, pode-se considerar um estudo maior acerca dos parâmetros do algoritmo genético e outras formas de seleção, além de outras opções de busca local como por exemplo a aplicação da heurística de *lin-kernighan* em busca de melhores soluções.

Referências

(Adam Lipowski, Dorota Lipowska) (2011). "Roulette-wheel selection via stochastic acceptance". *Physica A: Statistical Mechanics and its Applications*.