Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Bacharelado em Sistemas de Informação

FlyFood

Lucas Daniel Melo Gonçalves da Silva

Recife

abril de 2022

Resumo

Em uma sociedade que investe em otimização de rotas, tende a resolver vários problemas, tanto para aqueles que trabalham diretamente, quanto para os que trabalham nas áreas de pesquisas. A necessidade deste trabalho foi criar um programa capaz de resolver um problema de rotas em específico, consequentemente resolver o famoso *NP*-completo, o Problema do Caixeiro Viajante. A técnica usada nesse programa, que utiliza da linguagem python, foi o Algoritmo Genético, uma meta-heurística que tem como fundamentação a teoria da seleção natural de Charles Darwin e em outras técnicas da biologia evolutiva. E os resultados desse programa atendem até certo ponto o objetivo principal, e como será mostrado nas seções futuras, existem algoritmos capazes de resolver esse problema, mas que infelizmente não em tempo polinomial.

Palavras-chave: Algoritmo Genético. Caixeiro Viajante. FlyFood. Meta-heurística. *NP*-completo. Otimização de Rotas. Tempo Polinomial.

1. Introdução

1.1 Apresentação e Motivação

Em um futuro não tão distante, as cidades têm se desenvolvido e estão lotadas de prédios e construções. O combustível além de caro, está escasso, e por isso se vem pensando em outros meios de se locomover e de transportar produtos. As empresas têm buscado formas de otimizar as rotas de seus funcionários que ainda utilizam do combustível, para que não gastem muito, e tem utilizado drones para fazer entregas. Mas como é algo não tão potente como um caminhão, não podem carregar as entregas por muito tempo, e se for pensar em *delivery* de comidas, precisarão ser rápidos para que a qualidade do serviço continue boa.

O projeto do FlyFood é um clássico problema de otimização de rotas, que consiste em passar por todos os pontos de interesse, e retornar ao ponto de origem. A rota escolhida, e portanto, a melhor, deve ser a de menor distância, desta forma tornando esse problema similar ao clássico Problema do Caixeiro Viajante (PCV). A importância de resolver esse problema, é que ajudaria em muito pessoas que trabalham com entregas (ou outros serviços relacionados), aumentando a sua produtividade. Por consequência da resolução desse problema, traria uma nova visão em relação a otimização de rotas, e para problemas matemáticos e computacionais.

1.2 Formulação do problema

A dificuldade desse problema é achar todas as rotas possíveis e depois escolher a de menor custo - menor distância. A fim de compreensão, foram usadas 4 cidades, ou pontos de interesse, no problema do FlyFood, e se colocarmos num fatorial, teremos ***R(n) = n!/2*** rotas possíveis, 12 para $n = 4$ - por não se tratar de um ciclo direcionado, a distância de um percurso, usando como exemplo, ACDB será a mesma que um percurso BDCA, por isso o número de rotas é dividido por dois. Mas essa fórmula apenas nos diz o número total de rotas, e não como acha-las, por isso vamos usar uma Função Objeto que minimiza o custo das rotas, que será $min \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij}$, onde j e i são os pontos de interesses, c_{ij} é o custo de saida do ponto i ao ponto j, e o x_{ij} seria se o ponto i foi ao ponto j. É relativamente baixo o número de rotas, porém por se envolver num problema fatorial, crescerá exponencialmente na medida em que aumenta o valor

de n, tornando inviável uma solução polinomial (ou em um tempo considerado "satisfatório") para o problema.

Essa função está sujeita às seguintes restrições:

$\sum_{i=1,\, i\neq j}^{n} x_{ij} = 1,\ \forall j \in N \rightarrow$ chega apenas um caminho a cada ponto j.

$\sum_{j=1,\, j\neq i}^{n} x_{ij} = 1,\ \forall i \in N \rightarrow$ de cada ponto i sai apenas um caminho.

$\sum_{i\in S}^{n} \sum_{j\in S}^{n} x_{ij} \leq |S| - 1,\ \forall S \subset N \rightarrow$ elimina subrotas.

$x_{ij} \in \{0, 1\}\ \forall i, j \rightarrow$ não negativos.

1.3 Objetivos

O objetivo geral do trabalho é tentar resolver (mesmo que seja utópico) em um tempo polinomial, considerando que é inspirada num problema *NP*-completo. E os objetivos específicos serão: conseguir fazer um programa que seja excelente para valores pequenos, conseguir fazer com que esse mesmo programa tenha um resultado razoável para valores grandes e conseguir explicar de uma forma compreensível esse problema matemático, que até hoje não se tem uma solução polinomial.

1.4 Organização do trabalho

No capítulo **2**, serão tratadas as referências encontradas para a compreensão e resolução do problema. O capítulo **3**, será a seção onde mostrará outros trabalhos parecidos, como esse projeto do FlyFood, seus métodos e técnicas (evitando sempre de cometer plágio). Entrando no capítulo **4**, é onde os métodos, ferramentas e dados existentes, serão usados e apresentados. Já nos capítulos **5** e **6** serão (respectivamente): os experimentos e testes de resolução do problema, os resultados obtidos e explicar cada um deles, e a conclusão de todo o projeto, se foi ou não alcançado os objetivos e explicá-los.

2. Referencial Teórico

2.1 Classes de Complexidade

A classe de complexidade *NP* é definida como um conjunto onde estão todos os problemas de decisões, cujas soluções podem ser verificadas, mas não eficientemente achadas. Por isso o *nondeterministic polynomial*, da sigla *NP*. O termo não-determinístico, se refere a máquinas de Turing não-determinísticas, uma estratégia de formalizar matematicamente a ideia de um algoritmo de busca usando força bruta - um exemplo seria o algoritmo guloso.

Já os chamados problemas *NP*-completos, estão num subgrupo onde se encontram os problemas mais difíceis em *NP*, e ainda não se tem a certeza se existe ou não algum algoritmo polinomial que o resolva (o termo "completo", se refere à propriedade de ser capaz de simular tudo na mesma classe de complexidade). Caso exista um algoritmo capaz de resolver um problema *NP*-completo em tempo polinomial, então é garantido que todos os outros problemas de complexidade *NP*, também podem ser resolvidos. Então é possível notar, que esse projeto do FlyFood pertence à classe *NP*-completo, por saber que é possível de resolver, mas que atualmente não em tempo tratável.

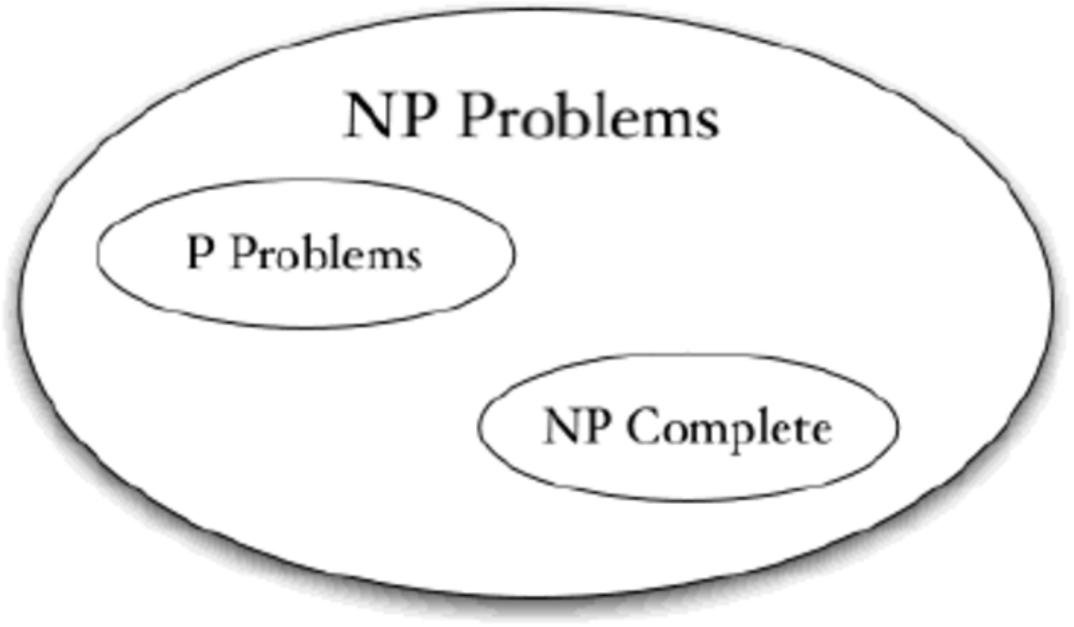

Figura 1. Diagrama das classes de complexidade

2.2 Complexidade de Algoritmos

É a quantidade de trabalho necessário para executar uma tarefa. Isto é medido em cima das funções fundamentais que o algoritmo é capaz de fazer (acesso, inserção, remoção, são exemplos mais comuns em computação), o volume de dados (quantidade de elementos a processar), e a maneira como ele chega no resultado - esta quantidade de trabalho está relacionada com o tempo, mas também pode referir-se à quantidade de memória necessária para garantir a execução.

Na figura a seguir, é mostrado as ordem de complexidade mais comuns e seus graus de 'agradabilidade', e como o problema do FlyFood utiliza de fatorial para encontrar todos os caminhos possíveis, a complexidade de seu algoritmo é *O(n!)*.

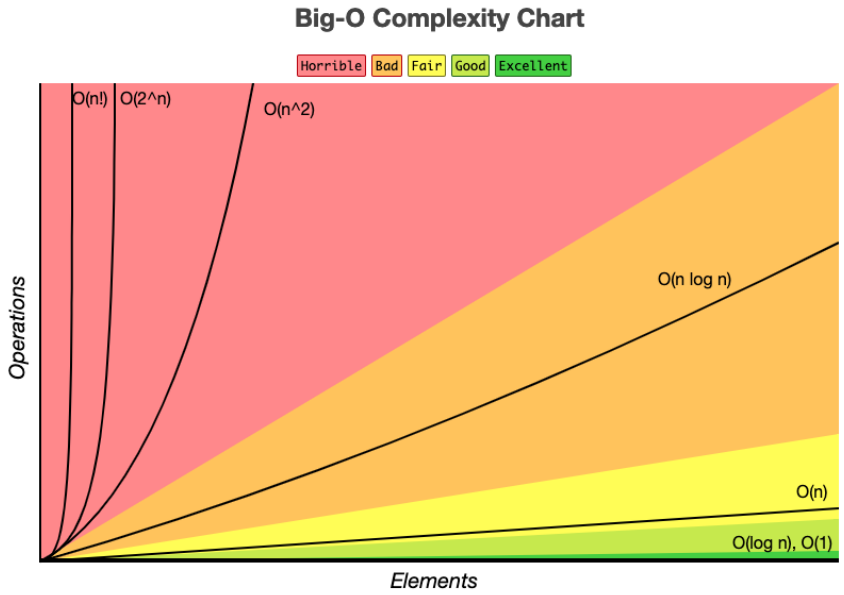

Figura 2. Gráfico das ordens de complexidades

2.3 Algoritmo do Vizinho mais Próximo

Os métodos heurísticos são algoritmos, que em pouco tempo conseguem obter soluções boas de problemas complexos, mas que não garantem uma solução ótima. Um método heurístico clássico para se aplicar nesse problema de rotas, a início, é o Algoritmo do Vizinho mais Próximo. Esse método trabalha usando as decisões de momento, o algoritmo vai sempre escolher a rota de menor custo, sem ter a informação de custos adiantes - essa heurística se enquadra nos padrões de um algoritmo guloso. Ao final do ciclo, resultará num custo total, mas o algoritmo não

garante se o caminho encontrado foi de fato uma escolha ótima, porém para pontos de interesses pequenos e para problemas que precisem serem resolvidos rapidamente, esse método é uma opção viável.

3.Trabalhos relacionados

3.1 Algoritmo Genético

Na Dissertação de Mestrado do Jean Gleison (Algoritmos de Solução para o Problema do Caixeiro Viajante com Passageiros e Quota, 2017), ele explica sobre vários algoritmos de solução para o problema do caixeiro viajante, e um deles é o Algoritmo Genético. Antes, será explicado alguns conceitos prévios, e depois a junção desse algoritmo com o problema do caixeiro viajante, incluindo o problema do FlyFood.

A meta-heurística é definida como conceitos utilizados para resolver um conjunto de problemas através de métodos heurísticos, que não possuem algoritmo estabelecido, empregue para encontrar soluções subótimas por intermédio da avaliação de soluções de qualidade com baixo custo computacional, bom desempenho, rapidez e simplicidade. O algoritmo genético é um desses métodos, e tem como fundamentação as premissas de Charles Darwin em que os indivíduos menos aptos (caminhos de maior custo) são desprezados e os mais aptos (caminhos de menor custo) selecionados. Nesse contexto, o problema do caixeiro viajante pretende descobrir o menor caminho de um número finito de vértices/pontos com distâncias entre se conhecidas.

Um algoritmo genético (AG) é um método de otimização baseado em mecanismos evolucionistas, que seguem a teoria da seleção natural e da sobrevivência de seres mais aptos, temas estes abordados por seu criador Charles Darwin. De acordo com essa teoria, quanto mais adaptado um ser é ao meio, maior a probabilidade de este sobreviver e de se reproduzir. Por saber que o algoritmo genético procura a solução ótima dentro de um determinado problema (Problema do Caixeiro Viajante), ou seja, a solução menos custosa, torna esse método interessante na hora formular uma resolução para o problema do FlyFood.

3.2 Algoritmo Memético (Algoritmo Híbrido)

No artigo produzido por Felipe Fernandes e Ingridy Barbalho (Algoritmo Memético Aplicado ao Problema do Caixeiro Viajante de uma Empresa Situada no Município de

Angicos/RN, 2016), eles aplicam uma meta-heurística famosa chamada Algoritmo Memético, que tem como base o Algoritmo Genético, e no artigo eles explicam como foi a implementação desse algoritmo para o problema deles, envolvendo um município do RN.

Em muitos casos, os algoritmos evolutivos (outra forma para se referir ao Algoritmo Genético) não são suficientes para encontrar a solução ótima. Pode se haver a importação de conhecimentos de domínios do problema de busca, quando eles existirem. Embora os algoritmos evolutivos tentam equilibrar a exploração e explotação - **exploração** seria a análise total de um determinado espaço de busca, e a **explotação** seria encontrar a melhor solução desse espaço de busca -, acabam sendo muito bons na exploração, mas não tão bons na explotação. É nesse momento que é feita a junção do Algoritmo Memético, e realizado algum processo de Busca Local no escopo do Algoritmo Genérico. As tecnologias híbridas em geral têm um desempenho melhor do que cada uma das tecnologias separadas, pois utilizam ambas de forma harmoniosa, combinando seus pontos fortes.

A Busca Local foi uma estratégia usada pelos autores do artigo, para aprimorar uma solução inicial intensificando a procura de uma solução final que representasse um ótimo local ou uma solução aceitável. Essa estratégia é frequentemente usada em heurísticas genéricas, em relação a problemas de otimização, e essa estratégia é baseada em uma estrutura de vizinhança. É recomendado para o problema do caixeiro viajante simétrico o algoritmo *Lin-Kernighan*, em que *k* arestas são intercambiadas de forma adaptativa - essa heurística é uma generalização de outros algoritmos, como o *2-opt*, *3-opt* e *k-opt*, que possuem a mesma ideia inicial. Envolve a troca de pares de subcaminhos para fazer um caminho novo. Esse algoritmo é adaptável e em cada etapa decide quantos caminhos entre os pontos (cidades) precisam ser trocados para encontrar um caminho mais curto. Para se ter uma noção, as instâncias do problema do caixeiro viajante de até um milhão de pontos, consegue retornar um caminho de pelo menos 2% do ótimo global.

Um outro termo importante é o conceito de *memes*, que seria uma unidade de transmissão cultural. Por não ser algo biológico, entra em contraste com a unidade de transmissão biológica, os genes. A ideia de *memes* é que a informação que é recebida como tendo alta utilidade ou popularidade é copiada pelos indivíduos no seu entorno. E aplicando no presente problema, o *meme* seria um caminho ótimo encontrado, e que será usado como referência para achar outros caminhos e conectá-los.

4.Metodologia

4.1 Aplicando o Algoritmo Genético

A melhor forma encontrada para resolver o problema do FlyFood, foi usar o algoritmo genético. O programa feito gera todos os caminhos possíveis, faz junções com o outros caminhos aceitáveis, realiza mutações e resulta num caminho bom - por ser uma heurística não se sabe se o caminho achado é ótimo, mas é bom para o momento.

Alguns conceitos prévios a serem explicados: **população** é o conjunto de indivíduos que estão sendo cogitados como solução, e que serão usados para criar um novo conjunto de indivíduos para análise. O **indivíduo** (pode ser chamado de cromossomo), é uma possível solução para o problema. **Genótipo**, no caso do problema do FlyFood, é cada ponto de interesse a ser visitado. **Cruzamento** (também conhecido como *Crossover* ou Recombinação) é quando são escolhidos dois indivíduos e troca trechos de genes entre si. **Mutação** é quando um ou mais genes de um indivíduo são alterados aleatoriamente. Já o **fenótipo** é designado a interação de todos os genes, que seria a rota.

Foram usadas as bibliotecas de *itertools* para fazer as permutações, *operator* para realizar uma operação em específica, *random* para executar certas funções aleatórias (como no caso da mutação) e *time* para marcar o custo de todo o algoritmo. Na classe de Aptidão ou *Fitness*, é onde irá ocorrer as análises de quais indivíduos (soluções) são boas, ou seja, o quão próximo o indivíduo está da solução desejada ou uma solução boa - para o projeto do FlyFood, essa aptidão está associada a menor rota.

No algoritmo foram usadas como dados: o tamanho da população como 100, 5% de mutações e foram feitas 50 gerações (iterações), que também é uma condição de parada.

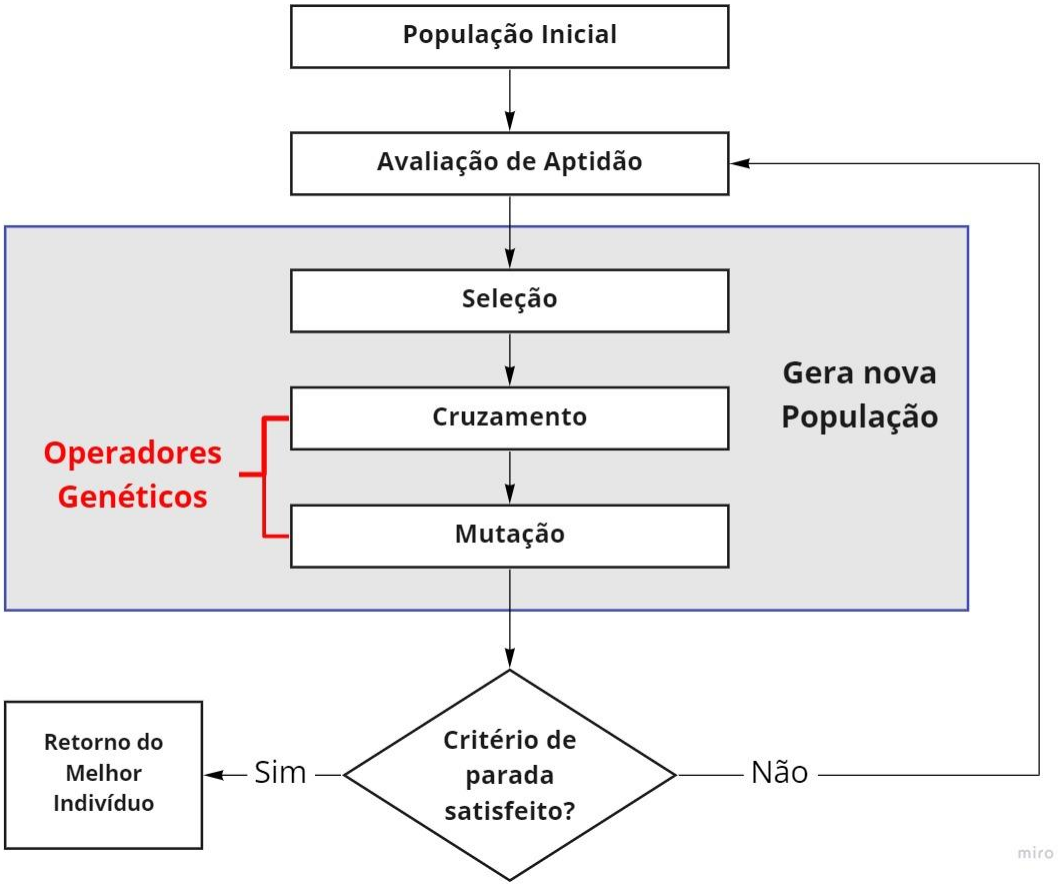

Figura 3. Diagrama básico de um Algoritmo Genético

5.Experimentos

5.1 Entradas

Para as entradas foram usadas sete matrizes, como podem ser exemplificadas na figura 3, que vão de 4 até 10 pontos de entrega (interesse). Nos espaços onde não tem pontos foram preenchidos com 0 - para que sejam contadas as distâncias dos pontos -, e essas entradas estão em formato *.txt*, e são acessadas/lidas usando o comando with open("entrada.txt", "r") as rotas no Python. Em cada matriz existe um ponto R, e como é o ponto de partida e chegada esse mesmo ponto será usado para encontrar os caminhos, por isso que no momento da impressão do caminho bom encontrado, é desconsiderado a repetição do ponto R. Será explicado na próxima subseção, mas o limite da quantidade de pontos a serem analisadas é de até dez porque aumentando o número de pontos, o programa não consegue resolver em um tempo aceitável.

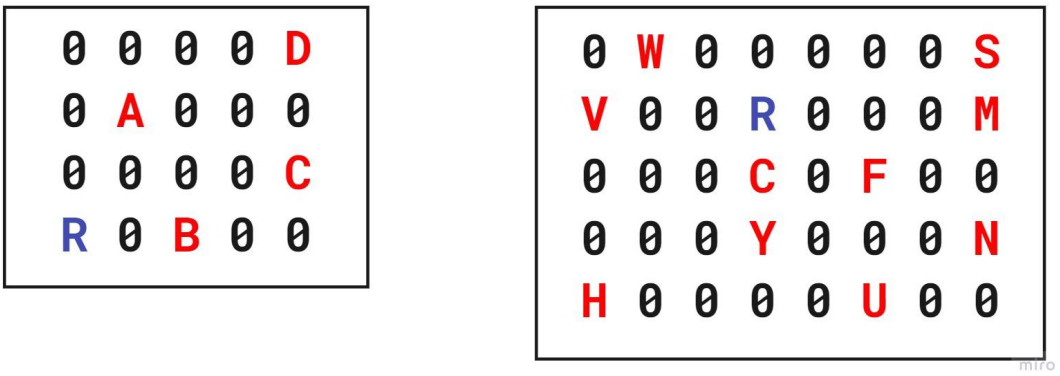

Figura 4. Matrizes de 4 e 10 pontos, respectivamente

5.2 Resultados

O programa feito conseguiu até 7 pontos retornar um caminho em um tempo agradável, a partir de 8 pontos era possível notar uma certa demora na impressão dos resultados. Para cada matriz foi rodado cinco vezes, a fim de tirar uma média aritmética dos resultados. Na figura 4 a seguir, a linha de **Tempo** é medida em segundos - se o programa levar muito tempo para dar a resposta, será convertido para uma unidade de medida mais compreensiva. Já na linha de **Custo** seriam os "dronômetros", ou seja, a distância total do percurso. E na última linha, **Melhor Sequência** indica qual foi o melhor caminho encontrado depois de achar a média.

O motivo de alguns custos serem maiores, em relação ao número de pontos, foi por causa do tamanho das matrizes. As matrizes não foram obrigatoriamente matrizes quadradas, elas são irregulares, como no caso da matriz 7x9 de 7 pontos que resulta num custo de 30,5, mas na matriz 5x7 de 8 pontos resulta num custo 29. É possível notar que com o aumento de um único ponto, o tempo levado para descobrir um caminho bom com 7 pontos para 8 pontos, é quase cinco vezes mais demorado.

Pontos / Resultados (%)	4	5	6	7	8	9	10
Tempo	0,39s	0,46s	0,96s	3,61s	20,15s	5min1s	44min41s
Custo	14	24	22	30,5	29	29,2	37,2
Melhor Sequência	B-C-D-A	E-B-U-G-S	I-S-L-E-D-C	P-V-B-H-C-A-D	A-G-N-S-T-B-V-I	I-Z-V-M-Q-B-H-E-U	V-W-F-C-H-U-Y-N-M-S

Figura 5. Tabela dos Resultados

5.3 Análise dos resultados

Com os resultados organizados, é possível perceber que o programa funciona bem para entradas pequenas, e a partir da matriz de 8 pontos começa a ter certa demora. A base para os experimentos foi usando 4 pontos, porque foi pedido a priori na formulação do problema, e o limite superior foi uma matriz de 10 pontos, pois foi o que levou um tempo considerado longo e ainda bom para análise dos resultados. Se fossem utilizados 11, 12 ou 13 pontos, levariam horas para se ter o retorno de uma única vez, e esse tempo levaria muito mais para se poder tirar a média aritmética. Por isso, se aplicarmos em um cenário real, não é viável utilizar esse algoritmo para entregas feitas a partir de 8 pontos.

6.Conclusão

O objetivo geral do trabalho era fazer um programa que pudesse gerar o melhor caminho para uma rota de entregas, em um tempo considerado agradável. Como foi visto, até certa quantidade de pontos é possível retornar em um ótimo tempo, passando de 8 pontos já se tem uma demora.

O programa feito não conseguiu de fato resolver o problema do FlyFood em um tempo polinomial com entradas maiores de 11 pontos, e como esse problema foi inspirado no Problema do Caixeiro Viajante, também não foi possível resolver esse *NP*-completo. Porém, como foi colocado em objetivos específicos, que seriam criar um programa ótimo para entradas pequenas e bom para entradas um pouco maiores, como também poder explicar mais detalhadamente nesse documento o PCV, foi possível conseguir atingir esses objetivos.

Para concluir esse trabalho, como foi mostrado em seções anteriores, existem algoritmos capazes de resolver esse problema de otimização de rotas, porém não em um tempo tratável, mas é possível que em algum ponto, com o avanço de pesquisas e métodos, que seja resolvido esse problema matemático.

Referências Bibliográficas

Associação Paranaense de Engenharia de Produção (2019). O Problema do caixeiro viajante através do algoritmo genético. Disponível em:

http://aprepro.org.br/conbrepro/2019/anais/arquivos/09302019_220914_5d92b20230a58.pdf. Acesso em 22 mar. 2022

Felipe Santana; Ingridy Barbalho (2016). Algoritmo Memético Aplicado ao Problema do Caixeiro Viajante de uma Empresa Situada no Município de Angicos/RN. Disponível em: https://siaiap32.univali.br/seer/index.php/acotb/article/view/10719/6051. Acesso em 23 mar. 2022

J.F. Porto da Silveira (2000). O Problema do Caixeiro Viajante. Disponível em: http://www.mat.ufrgs.br/~portosil/caixeiro.html. Acesso em 02 mar. 2022

Jean G. S. Silva (2017). Algoritmos de Solução para o Problema do Caixeiro Viajante com Passageiros e Quota. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/24207/1/JeanGleisonDeSantanaSilva_DISSERT.pdf. Acesso em 20 mar. 2022

Laira Vieira Toscani; Paulo A. S. Veloso (2012). Complexidade de Algoritmos - V13 - UFRGS.

Prof. Rafael Lima (2020). Gurobi (Aula 17): Implementação do Problema do Caixeiro Viajante. Disponível em: https://www.youtube.com/watch?v=FsIOKCOxrHQ. Acesso em 14 mar. 2022

String Fixer. NP-completude. Disponível em: https://stringfixer.com/pt/NP-complete. Acesso em 14 mar. 2022

UNIVESP (2017). Pesquisa Operacional II - Aula 07 - O problema do caixeiro viajante. Disponível em: https://www.youtube.com/watch?v=Doy6cBjb8uw. Acesso em 14 mar. 2022

UNIVESP (2017). Pesquisa Operacional II - Aula 08 - O problema do caixeiro viajante. Disponível em: https://www.youtube.com/watch?v=yI9bRgXbE1c. Acesso em 14 mar. 2022