Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Bacharelado em Sistemas de Informação

Flyfood: Uma abordagem acerca de algoritmos de roteamento e classes de problema aplicada a logística

Carlos Vinícius Martins da Silva

Recife

Outubro de 2021

Resumo

O roteamento de veículos é um tipo de problema onde: dado um veículo e *n* pontos de entrega, deve-se escolher uma rota de entrega com a menor distância a ser percorrida.Podemos afirmar que, o principal motivo para se solucionar esse tipo de problema seria o tempo. Já que, pressupõe-se que a menor distância a ser percorrida, exigirá menos tempo para ser percorrida também. Além disso, o gasto de combustível também pode ser citado como mais um motivo para que a resolução desse problema seja relevante. Afinal, seja combustível fóssil,elétrico ou de qualquer que seja sua matriz, é necessário que seu uso seja eficiente, para evitar custos desnecessários.

A solução que damos ao um problema é um algoritmo de força bruta. Este algoritmo reúne todas as possíveis rotas de entrega e checa uma a uma. Tal procedimento tem por objetivo, verificar qual das possíveis rotas de entrega faz com que o drone percorra a menor distância para completá-la. A geração de todas as rotas de entrega é feita através de um algoritmo recursivo, que consiste na obtenção da permutação dos *n* pontos de entrega.

Nós concluímos o objetivo deste trabalho com sucesso, obtivemos uma solução ótima para o problema. Porém, foi também constatado que tal solução tem um custo de tempo computacional muito elevado. Isso se dá devido ao número de possíveis rotas de entrega crescer de forma exponencial de acordo com o tamanho da entrada.

Palavras-chave: roteamento, algoritmo de força bruta, recursivo, permutações, exponencial.

1. Introdução

1.1 Apresentação e Motivação

O modelo de vendas utilizando entregas a domicílio, vem se popularizando rapidamente ao redor de todo mundo, até mesmo em uma jovem república como o Brasil podemos observar sinais do alastramento desse tipo de serviço, muito potencializado pelo surgimento da pandemia do novo coronavírus. Sabendo disto, este projeto acadêmico parte de um cenário fictício em que estaríamos no ano de 2025, onde o trânsito estaria caótico, e as empresas de delivery já não conseguiriam fazer entregas em tempo aceitável. Além disso, os custos com entregadores estariam muito altos devido a grande oferta de empregos. Então um ex-aluno do curso de bacharelado de sistemas de informação teria a ideia de criar uma empresa chamada FlyFood que teria como objetivo fazer entregas utilizando drones. Pela descrição do cenário de criação do nosso caso de pesquisa, podemos concluir que: o principal motivo para a criação da empresa seria a falta de velocidade nas entregas feitas convencionalmente. Isto nos leva a fazer o seguinte questionamento: como escolher a melhor rota de entrega visto que o entregador seria um drone?.

Devido à falta de um condutor humano, o drone, dado as coordenadas relacionadas aos pontos de entrega, deveria definir a melhor rota para realizar todas as entregas, e ao fim voltar ao ponto de partida. Um ponto a ser questionado seria: por que é importante que o drone escolha a melhor rota de entrega?. Para responder isso voltaremos ao principal motivo de criação da empresa: a velocidade de realização das entregas. Além disso, podemos citar mais um motivo para a formulação de um novo algoritmo: a economia de combustível fóssil, elétrico ou de qualquer tipo que este drone utilizaria.

1.2 Formulação do problema

O problema do FlyFood consiste em uma matriz de entrada, com *L* linhas e *M* colunas, que seria uma representação dos pontos da cidade. Essa matriz conteria, o

ponto de partida determinado pela letra *R*, os *N* pontos de entrega, que seriam representados por coordenadas dentro dessa matriz, tal que seria um ponto com coordenadas (*x*,*y*), e o restante dos pontos seriam os pontos que o drone poderia utilizar para se locomover até o destino das entregas. O drone só poderia se locomover de forma vertical e horizontal pelos pontos da matriz.

Tendo nosso ambiente formado, e seja *i* *a* posição do drone, podemos descrever o evento do drone se mover uma posição como i = i+1. O valor de *i* ao alcançar o ponto de entrega adjacente será o custo da solução entre dois pontos, mas como isso poderia ser descrito de forma sintética?. É nessa parte do problema que descreveremos a equação da distância entre dois pontos. Anteriormente dissemos que um ponto é representado por coordenadas, tal que um ponto *N* seria uma posição (*x*, *y*), imaginemos dois pontos de entrega, sendo eles *A* e *B*, onde *A* seria (x_1,y_1) e B seria (x_2,y_2). Como mencionado anteriormente nosso drone não poderá se locomover pelas diagonais da matriz, devido a este fato utilizaremos em nossa abordagem, uma das geometrias não euclidianas, conhecida como geometria do táxi. Denominaremos a distância entre dois pontos como *D*, portanto, a distância entre dois pontos pode ser determinada por $\boldsymbol{D = |x_2-x_1| + |y_2-y_1|}$. Elucidada a questão de como encontrar a distância entre dois pontos, fica mais fácil visualizarmos nosso próximo questionamento: qual seria a equação do custo da solução de um percurso completo?. Quando dizemos percurso completo nos referimos ao ato de realizar todas as entregas e voltar ao ponto de partida, ou seja, o somatório do custo de entrega de todos os pontos adjacentes. Então seja o custo total do percurso D_t , podemos dizer que $\boldsymbol{D_t} = \sum_{k=0}^{n} D$.

Definidas as equações de distância entre dois pontos e de custo de um percurso completo, cabe a nós acharmos a solução para o problema. Para isso iremos relembrar o motivo pelo qual foi criada a empresa: realizar entregas de forma mais rápida através do uso de drones. Dito isso, a solução para o problema seria achar

um circuito, tal qual o custo de solução fosse o menor possível, seja essa solução f(x), podemos dizer que: f(x) = argmin D_t .

1.3 Objetivos

O objetivo do trabalho é conceber uma solução para o roteamento dos drones, fazendo com que sigam pelo caminho com a menor distância possível.

O mesmo trabalho, tem como objetivos específicos:

- Analisar o custo dos possíveis algoritmos utilizados no problema de roteamento.
- Construir a habilidade de solucionar problemas reais utilizando algoritmos.
- Produzir material que possa servir de referencial teórico para futuras pesquisas acadêmicas.

1.4 Organização do trabalho

Neste tópico iremos discutir sobre como nosso trabalho está organizado. Na primeira seção está a nossa introdução. Nela é possível encontrar a apresentação e motivação, a formulação do problema, além dos objetivos. Na seção dois, é possível encontrar o referencial teórico. Nele nós discutimos sobre os assuntos que estão relacionados ao nosso problema, como classes de problemas e análise de algoritmos. A seção três, por sua vez, traz os trabalhos relacionados. Nesta seção, nós elencamos três trabalhos acadêmicos que tratam de problemas semelhantes ao nosso. Ao decorrer dos tópicos, nós descrevemos os trabalhos elencados e analisamos semelhanças e diferenças das abordagens que são feitas nestes trabalhos com o nosso. A quarta seção é chamada metodologia, nela nós descrevemos como foi feito nosso trabalho. A quinta seção é a de experimentos. Nesta seção, é feita a descrição dos experimentos realizados para provar que o algoritmo que nós propomos serve para o nosso objetivo. A sexta seção é a nossa conclusão. Nela nós discutimos sobre o nosso problema, damos uma breve explicação do que foi feito, além de apresentar nossos resultados obtidos.

2. Referencial Teórico

Iremos dar início ao nosso estudo sobre tempo computacional e análise de algoritmos. Abordaremos conceitos aos quais o nosso trabalho está relacionado. Falaremos sobre classes de problema, introduzindo as classes P, NP e NP completo, além de, redutibilidade, algoritmos de verificação, algoritmos determinísticos e não determinísticos. É importante destacar que o custo de tempo que um algoritmo leva para resolver um problema é calculado pelo número de operações básicas que ele executa. Então, para tratarmos de tais assuntos, é de fundamental importância o conceito de problemas resolvíveis em tempo polinomial, também abordaremos sobre o tema.

2.1 Tempo polinomial

Como citamos anteriormente, o custo de tempo que um algoritmo leva para resolver um problema é calculado pelo número de operações básicas que ele executa. Sabendo disso, iremos introduzir o conceito de tempo polinomial para a resolução de um problema. De acordo com Cormen et al(2009) é pertinente afirmar que os problemas de NP-completude são tratáveis por razões filosóficas, e não matemáticas. Segundo o próprio Cormen et al., (2009, p.768) “embora seja razoável considerar como intratável um problema que exige o tempo $\Theta(n^{100})$, um número bem pequeno de problemas práticos exige tempo da ordem de um polinômio de grau tão alto”. Por meio dessas duas citações, o autor deseja nos informar que: embora possa existir um problema com custo na ordem de n^{100} , o que seria um tempo impraticável por resultar em um número grande demais, geralmente este não se aplica na prática, pois, problemas práticos exigem um tempo muito menor. O que nos leva mais uma vez a consultar Cormen et al (2009, p.768) “Ainda que o melhor algoritmo atual para um problema tenha um tempo de execução de $\Theta(n^{100})$ é provável que um algoritmo com um tempo de execução muito melhor logo seja descoberto ”. Todas essas citações nos levam a tomar a seguinte conclusão: quando falamos em um problema que é resolvível em tempo polinomial, estamos considerando um tempo que segue um polinômio. Além disso, estamos considerando um polinômio de grau baixo, tal que resulte em um tempo praticável para a resolução do problema.

2.2 Algoritmos de verificação

Daremos início agora ao estudo dos algoritmos de verificação, eles são muito importantes para as definições das classes de problemas que introduziremos mais a frente. Como citamos anteriormente, um problema resolvível em tempo polinomial é aquele em que este tempo de execução segue um polinômio. Mas existem problemas em que a sua solução é resolvível em tempo polinomial, porém, a sua verificação não, e é para isso que servem os algoritmos de verificação. De acordo com Cormen et al (2009, p.775)" Definimos um algoritmo de verificação como um algoritmo de dois argumentos *A*, onde um argumento é uma cadeia de entrada comum *x* e o outro é uma cadeia binária *y* denominada certificado". Então, um algoritmo *A* verifica se , para qualquer entrada *x* que pertença ao problema, existe um certificado *y* que *A* pode usar para provar que qualquer *x* é encontrado em tempo polinomial.

2.3 Algoritmos determinísticos e não determinísticos

Segundo Cormen et al(2009,p.03)"Informalmente, um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída". Tendo este conceito introdutório em mente, iremos definir os conceitos que nos são pertinentes no momento.

Podemos definir um algoritmo determinístico como uma função matemática. Onde sua entrada de dados seria o domínio da nossa função, e a saída de dados seria o conjunto imagem dessa mesma função. Além disso, podemos definir o algoritmo em si como a lei de formação da função que estamos apresentando para o nosso estudo. Um algoritmo determinístico é exatamente como uma função, não importando a situação, uma saída resultará sempre o mesmo valor ou conjunto de valores dado um determinado valor ou conjunto de valores como entrada. Para ilustrar esta afirmação, façamos um exemplo utilizando uma função onde ***F(x) = 2x+1***. Suponhamos que *x* seja igual a 1, logo somos obrigados a concluir que *F(x)* = 3, desde que *x* seja igual a 1, *F(x)* resultará no valor 3. É importante também destacar que um algoritmo determinístico é um caso particular de um algoritmo não determinístico. Isso nos leva a concluir que podemos transformar um algoritmo determinístico em um algoritmo não determinístico de forma fácil.

Já um algoritmo não determinístico funciona de maneira contrária a um algoritmo determinístico como o próprio nome sugere. Em um algoritmo não determinístico uma saída pode resultar um valor ou conjunto de valores diferentes dado um determinado valor ou conjunto de valores como entrada. Para ilustrarmos isso,

voltemos ao exemplo com a função em que ***F(x) = 2x+1***. Em um algoritmo não determinístico, seja *x* igual a 1, *F(x)* pode resultar quaisquer valores mesmo que *x* sempre seja igual a 1. Diferente do que destacamos anteriormente, um algoritmo não determinístico não pode ser transformado em um algoritmo determinístico de forma tão fácil.

2.4 Classes de problemas

Agora que temos consolidados os conceitos de: algoritmos resolvíveis em tempo polinomial, algoritmos de verificação, algoritmos determinísticos e não determinísticos, podemos enfim introduzir as nossas classes de problemas. São elas as classes P, NP e NP completo esta que também pode ser abordada por alguns autores como NPC.

A classe P segundo Cormen et al (2009, p.765)"consiste nos problemas que podem ser resolvidos em tempo polinomial". Esta afirmação nos leva a considerar que: um problema que pertença a classe P pode ser resolvido em um tempo que segue um polinômio. Além disso, problemas incluídos na classe P são determinísticos, afinal todo polinômio é determinístico seguindo a nossa definição para o que é ser um algoritmo determinístico.

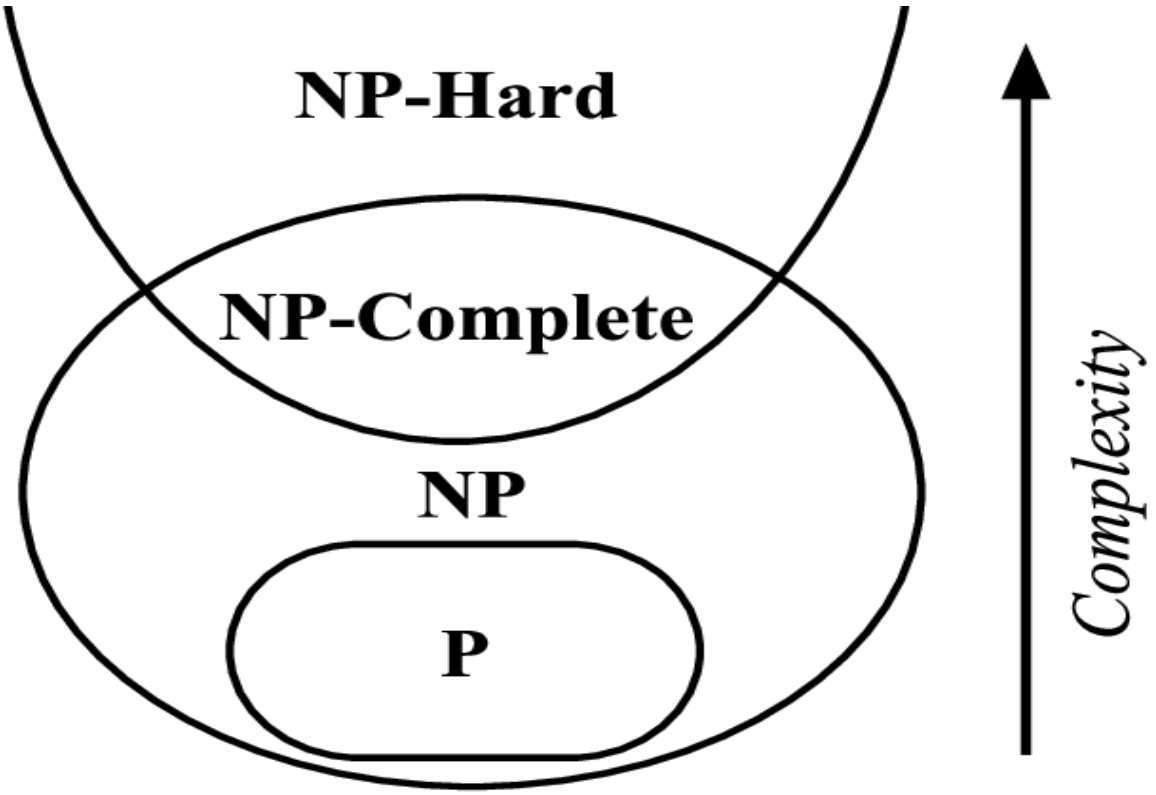
A classe NP de acordo com Cormen et al (2009) é composta por problemas que são verificáveis em tempo polinomial. Além disso, os problemas NP também são não determinísticos. Podemos dizer que $P \subseteq NP$, isso porque se um problema está em P, significa que podemos resolvê-lo sem nem mesmo precisar de uma verificação em tempo polinomial.

Segundo Cormen et al (2009, p.765)"Um problema está na classe NPC (e nos referimos a ele como um problema Np-completo) se ele está em NP e é tão difícil quanto qualquer problema em NP". Ou seja, problemas NP completo são não determinísticos e não podem ser resolvidos em tempo polinomial. Ainda de acordo com Cormen et al (2009) cientistas da computação acreditam que esses problemas sejam intratáveis devido a alta quantidade de problemas desse tipo já estudados, sem que ninguém tenha encontrado uma solução em tempo polinomial para nenhum deles. Até hoje acredita-se que $P \neq NP$, porém, seja um problema NP completo resolvido em tempo polinomial, então poderíamos afirmar que $P = NP$.

2.5 Classificação do Flyfood nas classes de problema.

O Flyfood, que é o nosso caso de estudo, encontra-se classificado como um problema NP hard. Um problema Np hard é aquele que é tão difícil quanto um problema NP completo. A figura abaixo demonstra com mais clareza esta afirmação.

Figura 1 - Diagrama de complexidade dos problemas computacionais



fonte: site researchgate.net[1]

3. Trabalhos relacionados

Neste tópico iremos abordar trabalhos relacionados, sendo concorrentes ou complementares ao nosso. Tendo nosso objetivo estipulado, elencamos três trabalhos acadêmicos para fazermos uma pequena análise. São eles: Algoritmo de busca dispersa aplicado ao problema clássico de roteamento de veículos (2007), Resolução do problema do caixeiro viajante no setor varejista usando heurística com previsão de trajetória e O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via Iterated Local Search e Genius(2010).

[1] Disponível em: <https://www.researchgate.net/figure/Diagram-of-intersection-among-classes-P-NP-NP-complete-and-NP-hard-problems_fig13_336890186?> . Acesso em: 16 out. 2021

3.1 Algoritmo de busca dispersa aplicado ao problema clássico de roteamento de veículos

Este trabalho acadêmico aborda o problema de roteamento de veículos e utilizam o conceito da meta-heurística busca dispersa para solucionar o mesmo. Para entendermos o funcionamento do algoritmo precisamos definir algumas nomenclaturas que são utilizadas no trabalho. *P* é um conjunto de soluções obtidas com o método gerador de soluções iniciais. *Psize* é o tamanho da população *P* de soluções iniciais. *Refset* é um conjunto de referência, enquanto *Pool* é um conjunto de soluções geradas pelo método da combinação. Já *b* é o tamanho do conjunto de referência, enquanto que b_1é o tamanho subconjunto do *Refset* correspondente a soluções de qualidade e b_2 é o tamanho do subconjunto do *Refset,* correspondente a soluções com diversidade. Já x^i é a $i^{ésima}$ solução do conjunto de referência, x^1 é a melhor solução em *Refset* e x^b a pior. Já *d*(*x,y*) é a distância entre a solução *x* e a solução *y*. Após definirmos as nossas nomenclaturas podemos seguir em frente e demonstrar as cinco etapas que compõem o algoritmo de busca dispersa. A primeira etapa consiste em gerar um conjunto *P* com *Psize* soluções, de onde será extraído um subconjunto que se denomina conjunto de referência *Refset.* Na segunda etapa é feita uma busca local para melhorar as *Psize* soluções inicialmente encontradas, se a solução analisada for inviável, o método tenta torná-la viável, já se for viável o método tenta melhorá-la. Na terceira etapa é construído ou atualizado o conjunto de referência *Refset*. A quarta etapa opera no conjunto de referência, e consiste em criar diferentes subconjuntos de soluções que serão utilizadas na etapa de combinação. A quinta e última etapa faz a combinação de soluções. Utilizando os subconjuntos gerados na etapa quatro, combina-se as soluções em cada subconjunto com o objetivo de encontrar novas soluções. É destacado pelo autor que esta etapa de combinação é um mecanismo específico para cada problema, uma vez que está diretamente relacionado com a representação da solução. O algoritmo básico de busca dispersa termina quando não existem novos elementos a serem combinados em *Refset*. Podemos destacar que o método utilizado pelos pesquisadores não busca a resposta ótima como nós, mas sim soluções de maior qualidade, com tempo aceitável. Ainda é destacado no artigo que a busca dispersa é um método evolutivo que combina soluções com o objetivo de criar novas soluções de melhor qualidade. No artigo, o autor não dá contexto ao leitor sobre classes de problemas e assuntos relacionados que precisam ser abordados para o entendimento destas classes, fato este que pode ser considerado uma lacuna no artigo. Fazendo desta forma com que o artigo seja muito prático, demonstra resultados de experimentos científicos para comparação de tempo entre algoritmos,

porém, sem dar um preâmbulo ao leitor do que seria tempo computacional e custo de algoritmos. O artigo conclui-se dizendo que as soluções encontradas pelo algoritmo de busca dispersa são muito próximas das melhores soluções reportadas na literatura para essas instâncias. Podemos definir então como a maior diferença entre nossos trabalhos, a abordagem do problema, enquanto nós buscamos a resposta perfeita o artigo busca uma resposta em tempo computacional aceitável.

3.2 Resolução do problema do caixeiro viajante no setor varejista usando heurística com previsão de trajetória

Neste artigo, o cenário abordado é muito parecido ao nosso, o cenário abordado pelo artigo é: uma simulação de rotas de entrega para reduzir distâncias percorridas em serviços de entrega. Para a realização do artigo, foi utilizado como unidade de pesquisa, uma microempresa que atua no ramo de comercialização e distribuição de água mineral do município de São Mateus - Espírito Santo(Brasil). Desta forma a problemática sobre a otimização de rotas de entrega no ramo de comercialização e distribuição de água mineral, foi formada por meio de entrevistas(questões fechadas), e reuniões junto a empresa que serviu de unidade de pesquisa. Após isso, os pesquisadores selecionaram quatro entregas, realizadas entre 13 e 15 de junho de 2019 em horário comercial. Para caracterizar as rotas utilizadas, utilizaram-se as seguintes informações: horário de partida e de chegada do entregador na empresa, pontos de entrega e caminhos utilizados pelo entregador. Para obtenção das rotas feitas pelo entregador, foi utilizada a ferramenta de roteirização do Google maps(GOOGLE, 2019). Posteriormente, os pesquisadores criaram uma matriz Origem-Destino para representar as distância entre todos os pontos de cada rota. Por fim, o problema do caixeiro viajante foi modelado no LINGO, implementando a heurística TSP. Todas as implementações foram feitas em um computador com as seguintes características: Intel Core I5 2.70 Gigahertz com 16 Gigabytes de RAM. Para solução do problema é empregada a heurística Traveling Salesman Problem(TSP) de Dantzig e Ramser (1959). Segundo o presente artigo os resultados permitiram identificar rotas otimizadas que obtiveram uma redução média de 6,1% das distâncias percorridas. O artigo não apresenta um estudo sobre tempo computacional e análise de algoritmos, fazendo com que um leitor que não seja um estudante da área de computação fique um pouco perdido sobre o assunto abordado. Na conclusão do artigo o autor deixa claro que o objetivo do estudo seria minimizar as distâncias percorridas. Consequentemente, como estamos falando de uma empresa real, é necessário que o algoritmo retorne esta

solução em um tempo aceitável. Portanto, este trabalho acadêmico não busca a resposta perfeita para o seu algoritmo, mas sim a resposta em tempo aceitável.

3.3 O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via Iterated local Search e Genius

Este trabalho acadêmico vem a estudar o problema de roteamento de veículos, com uma adição proposta por Min em 1989. Seria assim chamado o problema de: o problema de roteamento de veículos com coleta e entrega simultânea. A única coisa que é acrescentada neste problema é que no ato da entrega o veículo também faz uma coleta no ponto de entrega. O algoritmo proposto para resolver o problema é chamado de Genils. Este algoritmo combina o procedimentos heurísticos Iterated local search -ILS(Stützle e Hoos, 199), Variable Neighborhood Descent -VND(Hansen e Mladenoivic, 2001; Mladenovic e Hansen, 1997) e Genius (Gendreau et al., 1992). Portanto o objetivo do trabalho é encontrar uma resposta em tempo computacional aceitável e não uma resposta perfeita. No artigo não é feito nenhum estudo sobre tempo computacional ou análise de algoritmos, deixando assim o leitor somente ciente dos resultados em experimentos práticos medindo o tempo. No fim do estudo foi possível comprovar que o algoritmo Genius proposto é o que detém o maior número de melhores resultados, considerando os os três conjuntos consagrados de 72 problemas-teste da literatura. A combinação dos três procedimentos heurísticos anteriores fazendo assim o algoritmo Genils produz um comportamento interessante, que traz melhor desempenho em problemas de maior porte.

3.4 Conclusão geral sobre trabalhos relacionados

Como foi possível perceber, os trabalhos aqui citados têm alguns aspectos em comum. O primeiro deles: todos os trabalhos aqui elencados buscam uma resposta em tempo computacional aceitável e não uma resposta perfeita. O segundo deles: Não foi feita nenhuma explicação ao leitor sobre análise de algoritmos e tempo computacional. Nosso trabalho busca os dois aspectos em comum aos trabalhos elencados, nós buscamos uma resposta perfeita (tendo em mente que o nosso tem uma limitação de tempo) e damos ao leitor um preâmbulo sobre tempo computacional e análise de algoritmos.

4. Metodologia

Na formulação do nosso problema, nós definimos que o custo de uma rota de entrega seria: $\boldsymbol{D_t} = \sum_{k=0}^{n} D$. Se tivermos 4 pontos de entrega e sejam eles nomeados A, B, C e D, é compreensível dizer que uma rota de entrega pode ser ABCD. Sendo assim, a primeira coisa a fazer para resolver nosso problema é achar todas as permutações possíveis dados os pontos de entrega na nossa matriz inicial. Nós utilizamos um algoritmo de força bruta para resolver o problema, pois são feitas todas as permutações possíveis e calculado o custo de cada uma dessas permutações. Ao final do processo, o circuito cujo custo computacional for o menor, será definido como a melhor rota de entrega. Nas subseções que virão a seguir, iremos dividir as etapas do algoritmo em ordem sequencial para melhor entendimento.

4.1 Entrada e pontos de entrega

A entrada do problema, consiste em uma matriz *n* linhas por *m* colunas, onde cada ponto de entrega tem uma posição dentro dessa matriz. Os pontos de entrega são representados por letras do alfabeto, exceto a letra R que representa o ponto de partida. Para exemplificar a entrada do nosso problema podemos tomar a figura abaixo, onde a primeira linha consiste em 2 números inteiros que são *n* e *m* respectivamente, e as linhas seguintes são as linhas e colunas da matriz.

Figura 2 - Formato da entrada

```
4 5
0 0 0 0 D
0 A 0 0 0
0 0 0 0 C
R 0 B 0 0
```

Em seguida, para ilustrar melhor essa mesma entrada, demonstraremos sua respectiva matriz na imagem abaixo.

Figura 3 - Matriz da entrada exposta na figura 1

				D
	A			
				C
R		B		

Agora vamos demonstrar o algoritmo utilizado para obtenção dos pontos de entrega e de partida pelo algoritmo. Representaremos através de um pseudocódigo, onde A representa o arquivo de entrada estruturado de acordo com a figura 1. Nós iremos checar cada posição da matriz utilizando dois loops aninhados(um loop dentro do outro) para obter as informações que precisamos. Enquanto percorremos a matriz, iremos armazenar as informações que precisamos. Os pontos de entrega serão armazenados em uma lista. Além disso, iremos identificar as suas posições e a do ponto de partida na matriz e armazenar essas informações em um dicionário[2] para consultá-las quando necessário. Com todas essas informações pré-definidas, vamos ao nosso pseudocódigo.

```
Primeira-etapa(A):

    pontos_entrega = lista-vazia

    coordenadas = dicionário vazio

    loop i = 0 até n:

        loop j = 0 até m:

            se A[i][j] = 'R' faça:

                coordenadas['R'] = (i,j)

            porém se A[i][j] ≠ '0'  faça:

                adicione A[i][j] na lista pontos_entrega

                coordenadas[A[i][j]] = (i,j)
```

Perceba que a chave do dicionário sempre será o nome dado ao ponto de entrega ou ao restaurante(que no caso é R), e o seu conteúdo será suas r coordenadas *x* e *y.* Dessa forma podemos consultar essa informação facilmente, quando precisarmos calcular a distância entre os pontos. Também é importante enfatizar que a estrutura utilizada para armazenar as coordenadas de todos os pontos não precisa ser um dicionário. Nós utilizamos este meio devido a sua praticidade, mas qualquer

[2] Dicionários são coleções de itens não ordenados, onde uma chave é associada a um valor.

estrutura que replique um comportamento parecido ao destacado pode ser utilizado sem nenhum problema.

4.2 Geração de Permutações

Agora que conhecemos nossa entrada e pontos da nossa matriz, partiremos para a geração de permutações. A geração de permutações é parte crucial do nosso estudo, pois ela irá definir todas as possibilidades de rotas de entrega existentes. Sabemos que a permutação de um número seja ele *n* , é determinada por *n!*. Então sendo por exemplo 4 pontos de entrega como no nosso exemplo citado na subseção anterior, nós teremos 24 possíveis rotas de entrega. Para gerar essas permutações tão importantes, nós recorremos a um algoritmo recursivo. Não entraremos em detalhes sobre o que é um algoritmo recursivo. Resumidamente um algoritmo recursivo é aquele que divide um problema em problemas menores a serem resolvidos, para que no fim fique mais fácil de se resolver o problema inicial. Representaremos nosso algoritmo pelo pseudocódigo a seguir, onde a sequência de entrada A conterá os pontos de entrega dados na matriz de entrada. A saída consistirá em uma lista com todas as permutações geradas.

```
Permutar(A)
    se A.comprimento = 1 faça:
        retornar A
    lista = lista-vazia
    loop letra em A:
        aux = A.troca(letra, ‘’)
        adicione de forma iterativa em lista[letra + A para A em Permutar(aux)]
    retorne a lista
```

Perceba que o algoritmo recebe a sequência de caracteres, no caso do nosso exemplo da figura 1 ABCD. A cada iteração do loop ele seleciona uma letra que está em A. Na próxima linha do algoritmo, criamos uma variável chamada *aux*. Essa variável tem como referência a sequência A e troca a letra definida por um espaço vazio, ou seja, é como se nós tivéssemos apagado a letra da sequência. Na próxima linha do algoritmo será retornada as permutações possíveis de *aux*, e de forma iterativa será concatenada a letra que foi definida no loop com as permutações de

aux. O processo se repete até que o loop percorra todas as letras da sequência original.

4.3 Resolução do problema

Agora que temos todas as permutações, iremos resolver de fato o problema. É aqui que descobriremos qual a melhor rota para fazer todas as entregas. Vamos relembrar conceitos que discutimos na nossa formulação do problema. Lá nós determinamos que a distância entre dois pontos é dita por $\boldsymbol{D = |x_2-x_1| + |y_2-y_1|}$. E o custo total de uma rota de entrega é dito por $\boldsymbol{D_t} = \sum_{k=0}^{n} D$. Sendo assim, o nosso próximo pseudocódigo irá apenas utilizar um loop na lista de permutações geradas anteriormente e em cada item desta lista será utilizado outro loop que percorrerá todos os pontos de entrega. Após isso só precisamos resgatar as coordenadas de cada ponto, que foram armazenadas em um dicionário na primeira etapa do nosso algoritmo descrito na seção 4.1. Com essas informações, basta calcular as distâncias. Então nosso pseudocódigo de solução receberá uma lista de permutações e um dicionário com as coordenadas dos pontos de entrega.

Solução(lista-de-permutações, coordenadas):

contador = 0

tempo = 0

melhorTempo = 0

melhorCircuito = nulo

loop faça sequencia em lista-de-permutações até lista-depermutações.comprimento:

loop faça letra em sequencia:

se letra for a primeira letra da sequencia faça:

//Entre os primeiros colchetes está a chave e no segundo par de colchetes está o índice das coordenadas necessárias.

x_1 = coordenadas[R][0]

x_2 = coordenadas[letra][0]

y_1 = coordenadas[R][1]

y_2= coordenadas[letra][1]

tempo = tempo + (módulo de ($x_2 - x_1$) + módulo de ($y_2 - y_1$)

entregaAtual = letra

porém se letra não for a ultima letra da sequencia faça:

x_1= coordenadas[entregaAtual][0]

x_2= coordenadas[letra][0]

y_1= coordenadas[entregaAtual][1]

y_2= coordenadas[letra][1]

tempo = tempo + (módulo de ($x_2 - x_1$) + módulo de $y_2 - y_1$)

entregaAtual = letra

se letra for a ultima letra da sequencia faça:

x_1= coordenadas[entregaatual][0]

x_2= coordenadas[letra][0]

y_1= coordenadas[R][1]

y_2= coordenadas[entregaAtual][1]

tempo = tempo + (módulo de ($x_2 - x_1$) + módulo de($y_2 - y_1$)

x_1= coordenadas[entregaAtual][0]

x_2= coordenadas[letra][0]

y_1= coordenadas[R][1]

y_2= coordenadas[entregaAtual][1]

tempo = tempo + (módulo de ($x_2 - x_1$) + módulo de ($y_2 - y_1$)

se o contador for igual a 0 faça:

melhorTempo = tempo

melhorCircuito = sequencia

mas se tempo for menor que melhorTempo faça:

melhorTempo = tempo

melhorCircuito = sequencia

tempo = 0

contador = contador + 1

Perceba que em nosso pseudocódigo existem algumas variáveis, a primeira delas é o contador. O contador serve justamente para quando o loop passar pela primeira rota de entrega, definir o resultado dela como o melhor tempo e consequentemente seja a melhor rota de entrega. Isso porque quando começamos o algoritmo nós não temos uma referência para comparar o seu custo D_t com o custo atual. Na condicional que verifica se o tempo é menor que o melhor tempo gerado pela solução, definimos o melhor resultado que será o circuito que gera o menor tempo para a realização de todas as entregas a cada iteração. A distância entre dois pontos é definida por $\boldsymbol{D = |x_2-x_1| + |y_2-y_1|}$, a variável tempo é justamente a soma de todas essas distâncias em um circuito, calculada por $\boldsymbol{D_t = \sum_{k=0}^{n} D}$. E as variáveis *melhorTempo* e *melhorCircuito* definirão a resposta da nossa solução. Desde que não seja a primeira iteração, a cada circuito rodado pelo loop é verificado se o tempo gerado por ele é o menor tempo. Ao final do algoritmo, o circuito que é realizado com menor tempo $D_t$será escolhido como o melhor circuito, resolvendo assim o problema proposto.

5. Experimentos

Nesta seção, iremos discutir sobre como foram realizados os experimentos. Antes de tudo, vamos explicar sobre o nosso ambiente de pesquisa. O nosso código[3] foi escrito na linguagem de programação python. Utilizamos essa linguagem, devido a ser a utilizada convencionalmente em nosso curso de graduação. Porém, destacamos que não é um requisito obrigatório a utilização do python para replicar os experimentos realizados nesta seção. Todos os experimentos foram realizados em um computador com um processador AMD Ryzen 5 3.59GHZ e 16GB de memória RAM.

5.1 Entradas

No nosso estudo, nós elencamos 4 casos de teste, onde cada caso de teste continha um diferente número de pontos de entrega. Nesta subseção, iremos demonstrar as entradas e suas respectivas matrizes. Nós utilizamos em todos os casos de teste uma matriz de 5 linhas e 6 colunas.

O primeiro caso de teste continha três pontos de entrega, sendo eles: A, B e C. Resultando no seguinte arquivo da imagem abaixo. Lembrando que todas as entradas dos casos de teste são formatadas de acordo com a estrutura apresentada na figura 1.

Figura 4 - Arquivo de entrada do 1º caso de teste

Consequentemente, o arquivo produz a seguinte matriz de representação, ilustrada na figura abaixo.

[3] Código fonte disponível em https://github.com/vinicarlosss/Projeto_FlyFood/blob/main/flyfood.py .

Figura 5 - Matriz de representação do 1º caso de teste

A	0	0	0	0	0
0	0	0	0	0	R
B	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	C

Em nosso segundo caso de teste, utilizamos seis pontos de entrega, sendo eles: A, B, C, D, E e F. A imagem abaixo demonstra o arquivo de entrada para esse caso de teste.

Figura 6 - Arquivo de entrada do 2º caso de teste

```
5 6
A 0 0 0 0 F
0 0 0 0 0 R
B 0 E 0 0 0
0 0 0 0 0 0
0 D 0 0 0 C
```

Este arquivo consequentemente, gera uma matriz de ilustração, que é apresentada na figura 7 logo abaixo.

Figura 7 - Matriz de representação do 2º caso de teste

A	0	0	0	0	F
0	0	0	0	0	R
B	0	E	0	0	0
0	0	0	0	0	0
0	D	0	0	0	C

Nosso terceiro caso de teste foi realizado com oito pontos de entrega, sendo eles: A, B, C, D, E, F, G, H. A figura 8 irá demonstrar o arquivo de entrada contendo todos os casos de teste e o ponto de partida, representado pela letra R.

Figura 8 - Arquivo de entrada do 3º caso de teste

```
5 6
A 0 0 0 0 F
0 0 0 0 0 R
B 0 E 0 H 0
0 0 G 0 0 0
0 D 0 0 0 C
```

O caso de teste, nos trouxe a matriz de ilustração que será demonstrada através da figura 9, logo abaixo.

Figura 9 - Matriz de representação do 3º caso de teste

A	0	0	0	0	F
0	0	0	0	0	R
B	0	E	0	H	0
0	0	G	0	0	0
0	D	0	0	0	C

O nosso quarto e último caso de teste contou com dez pontos de entrega, sendo eles: A, B, C, D, E, F, G, H, I, J. Segue na figura 10, o arquivo de entrada para este caso de teste.

Figura 10 - Arquivo de entrada do 4º caso de teste

```
5 6
A 0 0 0 0 F
0 J 0 0 0 R
B 0 E 0 H 0
I 0 G 0 0 0
0 D 0 0 0 C
```

O último caso de teste gerou a seguinte matriz de ilustração, apresentada na figura 11, logo abaixo.

Figura 11 - Matriz de representação do 4º caso de teste

A	0	0	0	0	F
0	J	0	0	0	R
B	0	E	0	H	0
I	0	G	0	0	0
0	D	0	0	0	C

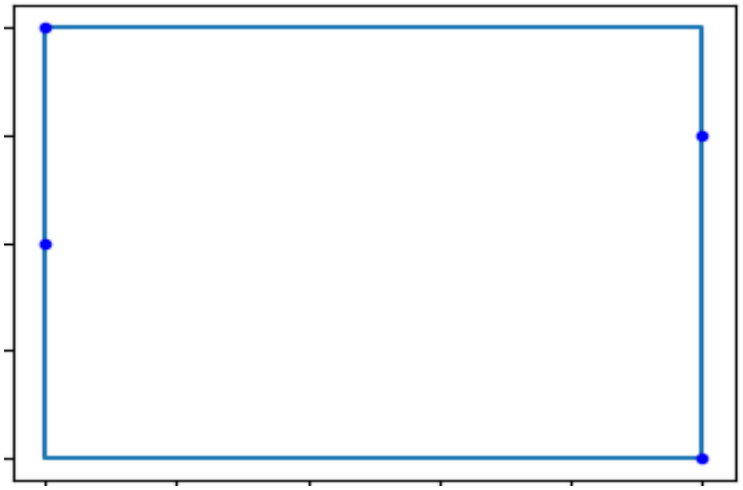
Com isso, nós temos todas as entradas que utilizamos em nossos casos de testes para os experimentos realizados neste trabalho. Agora podemos discursar sobre o experimento em si, o que faremos na próxima seção.

5.2 Resultados

Nesta seção, iremos abordar os resultados obtidos em nossa pesquisa. Para isso, nós executamos o programa a fim de medir o tempo gasto para resolver o problema nos quatro casos de teste. Para medir o tempo, utilizamos uma biblioteca da linguagem python chamada time. Utilizamos essa biblioteca, devido a mesma dispor de um método em que podemos marcar o início e o fim do programa. Assim, só é preciso diminuir o tempo marcado no fim do programa pelo tempo marcado no início, para descobrir quanto tempo foi gasto para resolver o problema.

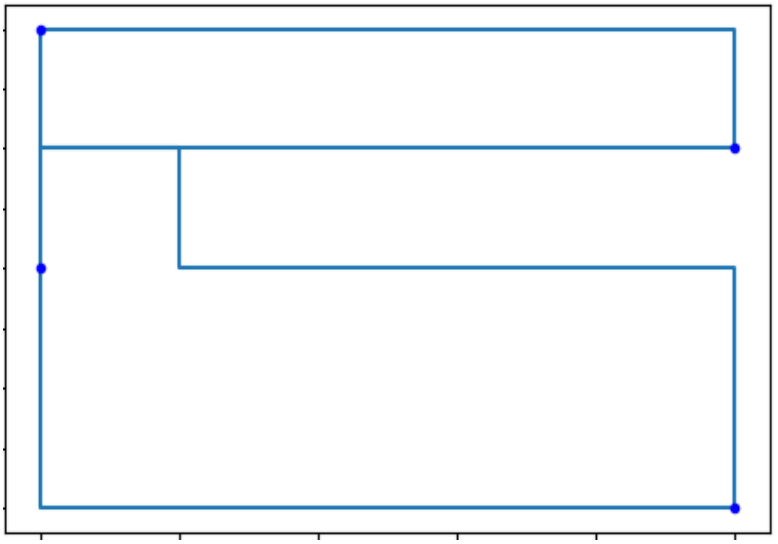
No primeiro caso de teste, que utilizou três pontos de entrega, o algoritmo foi executado em 0,001 segundos e o melhor circuito de entrega foi ABC, com uma distância total $D_t = 18$. Em seguida, iremos mostrar uma figura com o plot dos pontos de entrega ligados por arestas, para demonstrar a melhor rota encontrada pelo algoritmo.

Figura 12 - Plot da rota de entrega ótima do 1º caso de teste



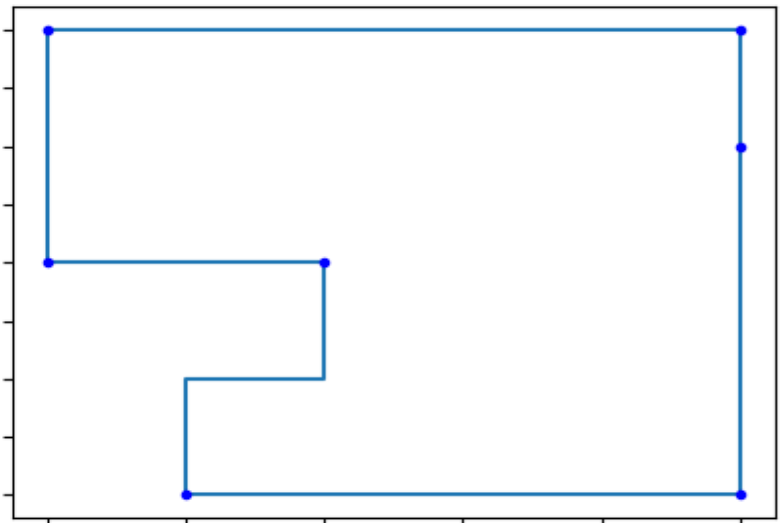
Agora iremos apresentar uma rota que não retorna a melhor resposta para o problema a fim de comparar resultados. Escolhemos a rota BCA, e demonstraremos a diferença de distância através da imagem abaixo. Perceba que, o número de espaços que o drone precisa se locomover para completar a rota é maior.

Figura 13 - Plot da rota de entrega que não retorna a resposta ótima do 1º caso de teste



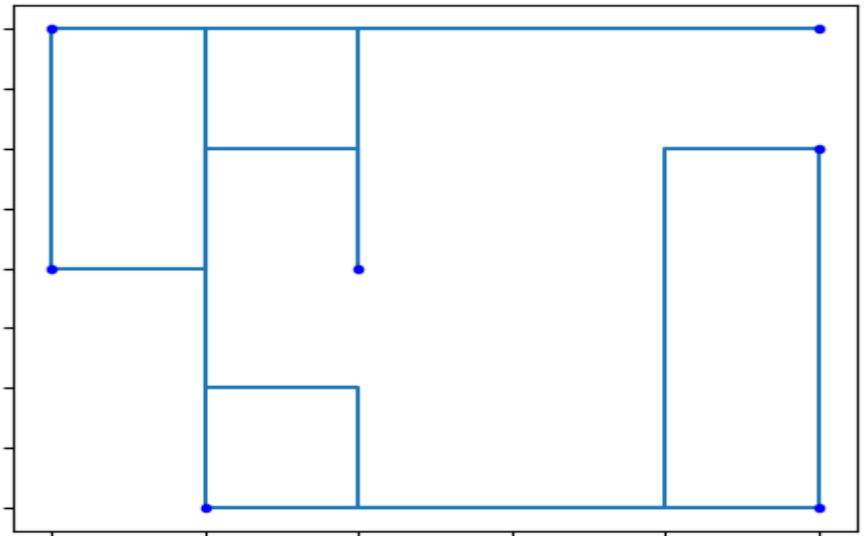
Agora iremos apresentar os resultados obtidos com o segundo caso de teste, utilizando 6 pontos de entrega. Neste caso de teste, foi possível observar o programa sendo executado em um tempo de 0,024 segundos. O programa nos retornou a rota FABEDC como a melhor rota de entrega com uma distância $D_t = 20$. A imagem abaixo, ilustra a melhor rota de entrega para o problema.

Figura 14 - Plot da rota de entrega ótima do 2º caso de teste



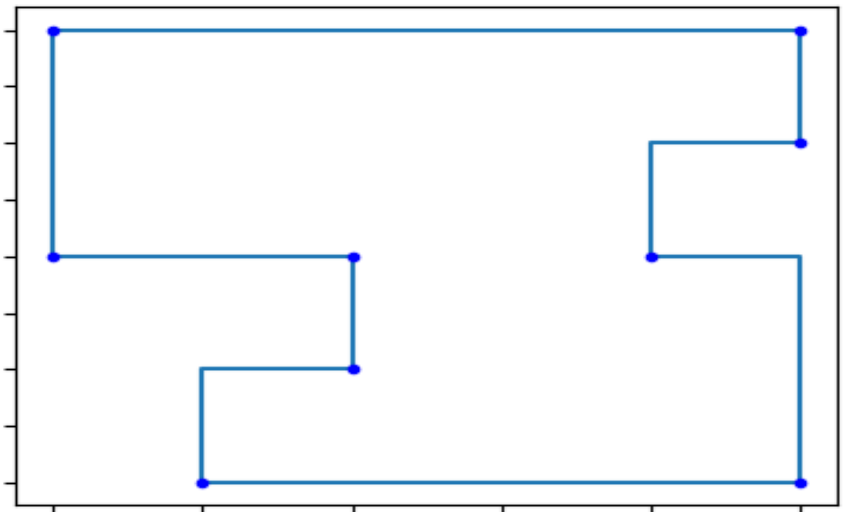
Com a imagem abaixo, será possível visualizar uma rota que não retorna a resposta perfeita para o problema. Assim, será possível visualizar a diferença de resultados, a rota escolhida foi: CABFED.

Figura 15 - Plot da rota de entrega que não retorna a resposta ótima do 2º caso de teste



Agora vamos discutir o terceiro caso de teste, que utiliza 8 pontos de entrega. Neste caso de teste, o programa foi executado em 1,578 segundos, já sendo possível notar um aumento significativo no tempo de execução(mais de 1 segundo). A melhor rota escolhida para o problema neste caso foi: FABEGDCH, com uma distância calculada $D_t = 22$. A figura abaixo mostrará em detalhes o caminho percorrido pelo drone.

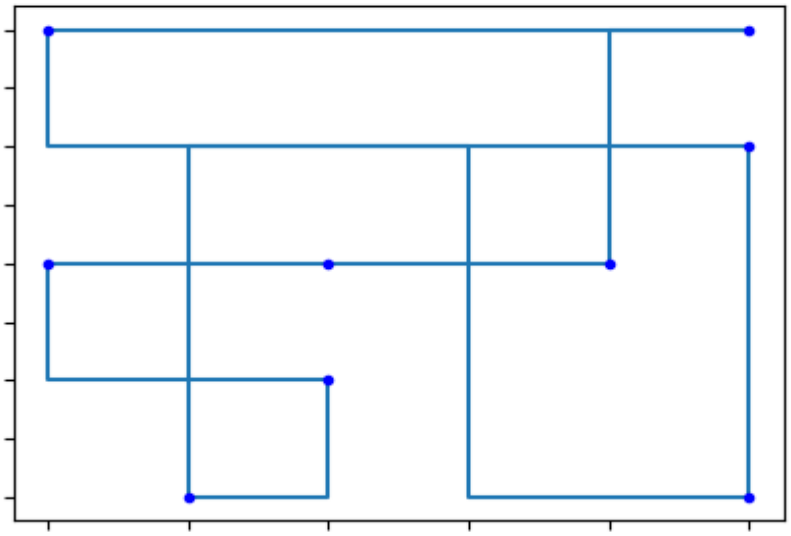
Figura 16 - Plot da rota de entrega ótima do 3º caso de teste



Como nos casos de teste anteriores, escolhemos uma rota que não retorna o melhor resultado. Assim, será possível demonstrar que o caminho que o drone percorre nesse percurso é muito maior. A rota de entrega escolhida foi: CDBAEGHF, que teve

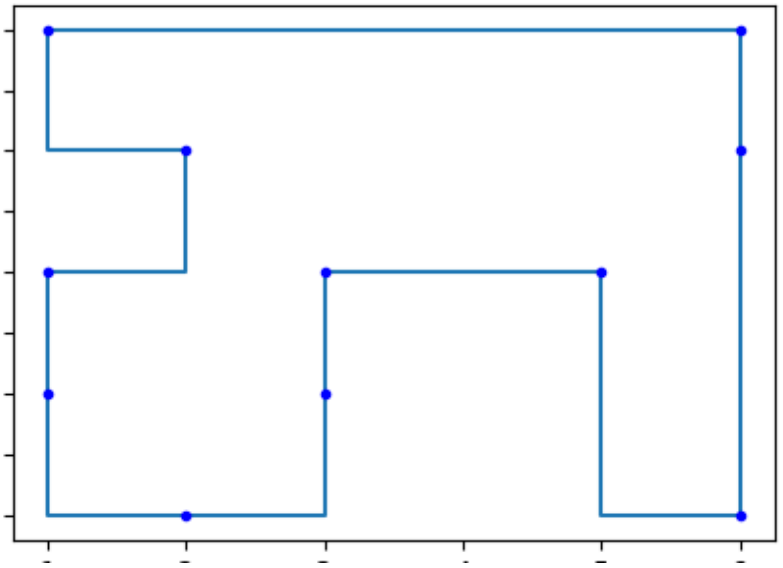
uma distância $D_t = 24$. A locomoção do drone pode ser visualizada na próxima imagem.

Figura 17 - Plot da rota de entrega que não retorna a resposta ótima do 3º caso de teste



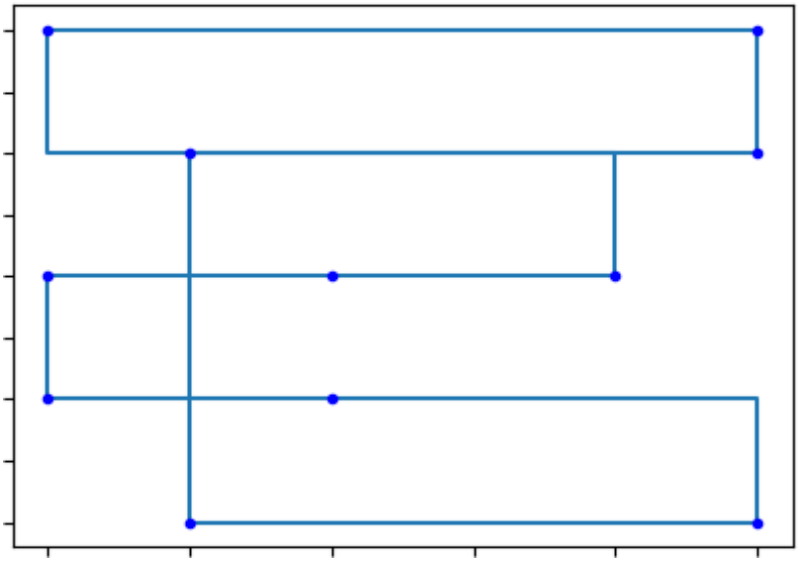
Chegando ao fim do nosso experimento, o quarto caso de teste utiliza 10 pontos de entrega. Neste caso, pode ser observado um tempo de 171,13 segundos para execução do programa, ou seja, aproximadamente 2 minutos e 51 segundos de execução. Como melhor rota, foi escolhida FAJBIDGEHC, com uma distância $D_t = 24$. A figura abaixo, ilustra o caminho percorrido pelo drone para realizar a entrega.

Figura 18 - Plot da rota de entrega ótima do 4º caso de teste



A figura abaixo, irá representar uma rota que não retorna a resposta ótima para o nosso caso de teste. Para ilustrar isso, escolhemos a rota FAJHEBIGCD, onde será possível notar que a rota feita pelo drone é maior.

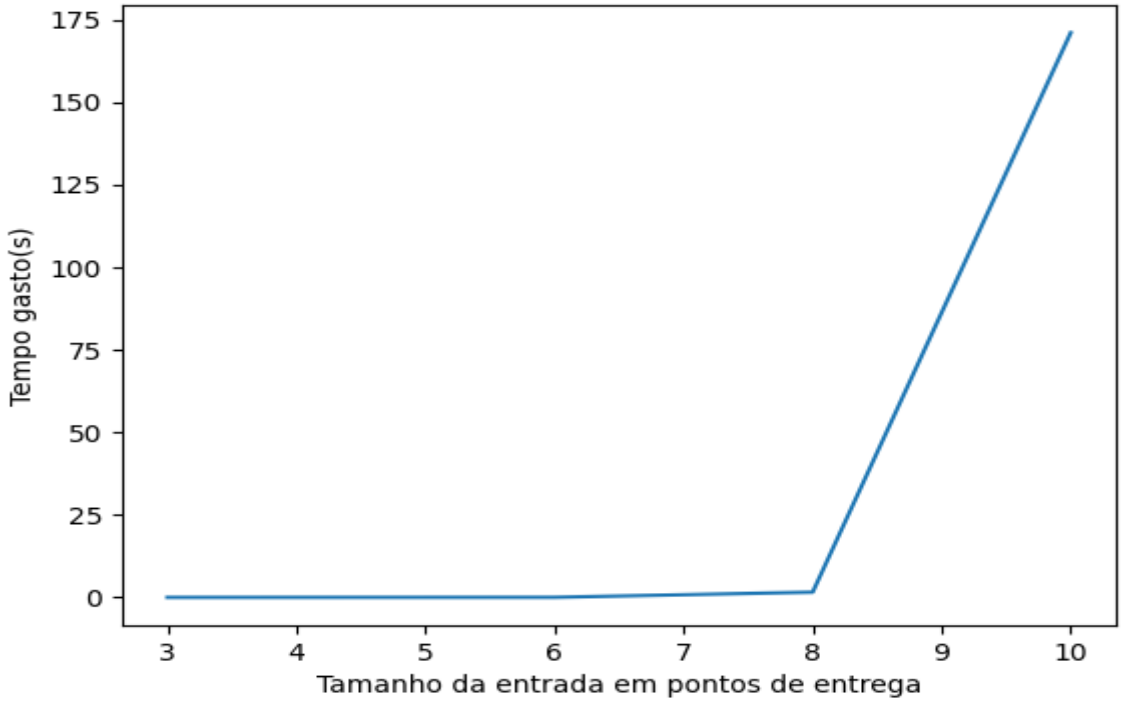
Figura 19 - Plot da rota de entrega que não retorna a resposta ótima do 4º caso de teste



Observados todos os resultados, podemos agora plotar um gráfico, que nos permite visualizar a relação entre tempo gasto de acordo com os pontos de entrega. Iremos demonstrar o gráfico na figura 20. Perceba que no gráfico o tempo gasto fica muito

próximo a zero até o programa ter que lidar com 8 pontos de entrega, após isso, a distância é mais perceptível.

Figura 20 - Gráfico de relação tempo gasto por pontos de entrega



5.3 Análise de resultados

Após todos os resultados, podemos concluir que o objetivo deste trabalho foi concluído com sucesso. Conseguimos apresentar um algoritmo que encontra a resposta ótima, dado os pontos de entrega. O único problema deste algoritmo é que ele não é viável para aplicações práticas. Pelo menos, não aquelas em que seja necessário calcular casos onde existam mais de 8 pontos de entrega. Foi possível analisar que, até 8 pontos de entrega o tempo gasto para calcular a menor rota ficou muito próximo de 0 segundos. Porém, a partir disso o tempo de execução com 10 pontos de entrega, por exemplo, subiu para mais de 2 minutos. No cenário do Flyfood, por exemplo, se o drone tivesse 10 pontos de entregas para serem realizadas, ele teria de esperar cerca de 2 minutos e 51 segundos para poder partir ao seu destino. Deixando um pouco os pontos de entrega de lado, se restringirmos o espaço de visão em quaisquer pontos do gráfico, é possível visualizar que o crescimento do tempo gasto é exponencial. O nosso algoritmo retorna a resposta

ótima, isso porque ele reúne todos os caminhos possíveis e verifica cada um deles. Este procedimento é realizado a fim de encontrar o circuito que retorne a menor distância D_t. Tal procedimento é chamado algoritmo de força bruta. Mas por que o nosso algoritmo é tão custoso? Para decidir, qual a rota de entrega percorre a menor distância é um problema fácil, apenas de comparação. Nós apenas precisamos comparar a rota que estamos analisando agora com a rota que analisamos antes, e verificar qual rota percorre a menor distância, isto é feito em tempo polinomial. O problema é que a obtenção das permutações que representam as possíveis rotas de entrega leva tempo exponencial para ser obtida. O trabalho do nosso algoritmo é procurar neste espaço enorme, a rota que percorre a menor distância, e é por isso que nossa solução é tão custosa. Para finalizar, podemos dizer que não solucionamos o problema em sua totalidade, pois, o algoritmo só é viável em problemas com poucos pontos de entrega. Porém, à medida que o número de pontos de entrega vai aumentando, não é mais viável utilizar este algoritmo.

6. Conclusão

O problema de roteamento de veículos está presente em vários aspectos de nossa economia. Seja por motivo de, otimização do tempo gasto ou eficiência de gasto de combustível, é de suma importância, que este problema seja pensado na hora de traçar uma estratégia para a logística de uma atividade. Isso pois, ao otimizar a rota de entrega, o tempo gasto para realizar o serviço será bem menor. Além de, os gastos com combustível também serem reduzidos, trazendo benefícios econômicos na execução da atividade. Este trabalho acadêmico, vem com o objetivo de alcançar um algoritmo que otimize a distância percorrida por um veículo dado a ele *n* pontos de entrega.

A fim de obter um algoritmo que retornasse a resposta com a melhor rota de entrega, foram realizadas pesquisas na bibliografia já existente, em livros, artigos e trabalhos acadêmicos. Ao fim da pesquisa, percebemos que a resposta ótima viria a partir de um algoritmo de força bruta, algoritmo esse que reúne todas as respostas possíveis e checa uma por uma até encontrar a melhor resposta. Todos os

experimentos realizados, foram a fim de medir o tempo computacional gasto de acordo com a quantidade de pontos de entrega que fossem fornecidos ao programa.

Ao fim de nosso estudo, podemos dizer que o objetivo deste trabalho acadêmico foi alcançado em sua totalidade. Pudemos provar que o algoritmo de força bruta retorna a resposta ótima, para qualquer circuito dado *n* pontos de entrega. Porém, à medida que aumentamos o número de pontos de entrega, podemos perceber um crescimento exponencial no tempo gasto pelo algoritmo para resolver o problema. Isso pois, como pudemos verificar em pesquisas na bibliografia existente,este problema é tido como NP-Hard. Isso se dá pois, para encontrar todas as rotas possíveis de entrega, nós precisamos fazer a permutação dos pontos de entrega. Esse vetor contendo todas as rotas de entrega, cresce exponencialmente de acordo com o número de pontos de entrega. Nosso algoritmo precisa fazer uma busca dentro desse vetor enorme, para encontrar a rota com a menor distância, e por isso demoramos um tempo tão alto para resolver o problema. Dito isso, podemos afirmar que seu uso é inviável para aplicações práticas que forneçam um número de pontos de entrega maior que oito. Esta é a quantidade de pontos de entrega limite que provamos, através de nossos experimentos, que dão uma resposta em tempo aceitável para aplicações práticas.

Referências Bibliográficas

CORMEN, Thomas.H *et al*. **Algoritmos:** teoria e prática. 3ª edição. Rio de Janeiro: Elsevier, 2012.

GONÇALVES, Wellington *et al*. **RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE NO SETOR VAREJISTA USANDO HEURÍSTICA COM PREVISÃO DE TRAJETÓRIA.** Revista desafios, v. 7, n. 3, 2020.

MINE**,** Marcio T *et al***. O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via Iterated Local Search e GENIUS.** Transportes: v. XVIII, n. 3, p. 60-71, setembro de 2010.

SOSA, Nélida *et al*. **ALGORITMO DE BUSCA DISPERSA APLICADO AO PROBLEMA CLÁSSICO DE ROTEAMENTO DE VEÍCULOS.** Pesquisa Operacional, v.27, n.2, p.293-310, Maio a Agosto de 2007**.**