



**Universidade Federal Rural de Pernambuco**

**Departamento de Estatística e Informática**

**Bacharelado em Sistemas de Informação**

## **FLYFOOD**

**José Carlos Eliodoro de Santana**

**Recife**

**12 de 2022**

# Resumo

Este trabalho tem como objetivo solucionar o problema de trânsito caótico nas cidades, especificamente no que diz respeito às entregas de comida. O tráfego de veículos nas grandes cidades pode ser um obstáculo para as empresas de delivery, pois, a velocidade de entrega é afetada. Para resolver esse problema, o presente trabalho propõe a utilização de drones para realização das entregas, utilizando um algoritmo de Força Bruta. O algoritmo tem como objetivo encontrar a rota mais curta para o drone realizar as entregas, levando em consideração os limites de voo dos drones e a menor distância entre os pontos da matriz que representam os pontos da cidade. Através da implementação desse algoritmo, foi possível concluir que ele é eficiente na solução desse problema, permitindo ao drone chegar ao destino de entrega mais rapidamente. Além disso, a utilização de drones para as entregas traz uma série de vantagens, como a redução dos custos de transporte de mercadorias e a agilidade na realização das entregas, pois, não precisam depender do tráfego de veículos.

# 1. Introdução

## 1.1 Apresentação e Motivação

Com o crescimento exacerbado das cidades por inúmeros fatores, observa-se que os sistemas de distribuição inseridos em tal ambiente, estão em maioria decadentes, o que fez com que os fornecedores de serviços tendessem a enfrentar diversos desafios no seu dia-a-dia, como incertezas relacionadas com a entrega de mercadorias, principalmente com o alto custo da mão-de-obra humana, o que os levou a explorar implementações do uso de meios alternativos para solucionar o problema.

Tendo em vista esses fatos a empresa do ramo alimentício Flyfood, desenvolveu uma maneira de realizar entregas com a utilização de drones, o que seria uma solução viável contanto que os drones possuísem um algoritmo de roteamento eficiente no qual os mesmos pudessem operar utilizando a melhor rota possível, para evitar desperdícios de cargas da bateria, enfim reduzindo o tempo gasto para entregas e por tanto aumentado a eficiência das entregas.

## 1.2 Formulação do problema

O problema pode ser formalizado matematicamente como um problema de otimização de rotas.

- **Entrada:** matriz de entrada será a informada com os pontos de origem e os de entrega.
- **Ponto vazio:** Representado por “0” é um ponto no qual nada será constado nele.
- **Ponto de partida:** Representado por “R” é um ponto no qual o drone deve começar é terminar o trajeto.
- **Pontos de entrega:** Os pontos de entrega serão onde o drone iram realizar as entregas dos pedidos representador por qualquer elemento diferente de “0” e “R”.

- **Distância entre pontos:** Será a distância medida em dronômetro.
- **Dronômetro:** Será a unidade de medida utilizada para percorrer a menor distancia de um ponto a para um ponto b em uma matriz  $M_{x,y}$  sem percorrer pela diagonal. Tem-se a distância representado por  $d(a, b) = |x_b - x_a| + |y_b - y_a|$ .

Com todos os conteitos iniciais podemos formular a resolução do problema, inicialmente calculando a distância entre todos os pontos de uma rota, porem a função  $d(a, b) = |x_b - x_a| + |y_b - y_a|$ , soma apenas a distância entre um único ponto, porém, é necessário uma solução que some toda a rota. Frente a isso sendo temos a função  $f(d) = \sum d$ . como buscamos o menor caminho dentre esses para realizar todas as entregas, temos que calcular o valor mínimo que retorna por essa função, e temos a função  $\text{argmin}(f(d))$ .

## 1.3 Objetivos

Este trabalho tinha como objetivo principal desenvolver um algoritmo que pudesse calcular a menor distância entre os pontos de uma matriz, representando pontos em uma cidade, levando em consideração os limites impostos para o movimento dos drones. O objetivo era encontrar uma forma de otimizar o transporte de mercadorias, reduzindo os custos envolvidos.

Além disso, serão propostas possíveis melhorias e aplicações futuras do algoritmo desenvolvido. Espera-se que este trabalho possa contribuir para a otimização do transporte de mercadorias, proporcionando benefícios econômicos e ambientais.

## 2. Referencial Teórico

Nesta seção, serão apresentadas as referências utilizadas para a elaboração do trabalho, a fim de proporcionar uma melhor compreensão dos temas abordados.

## 2.1 Complexidade de Algoritmo

O custo computacional é definido como a quantidade de recursos, tais como o tempo de processamento e espaço de armazenamento da máquina, o que faz com que um mesmo algoritmo tenha um desempenho diferente em máquinas com hardwares diferentes.

## 2.2 Classes de problemas

Na Teoria da Complexidade Computacional, uma Classe de problema é um conjunto de problemas relacionados aos recursos computacionais baseados em complexidade, das quais podemos dividi-los em quatro classes de problemas que seriam P, NP, NP-completo e NP-difícil.

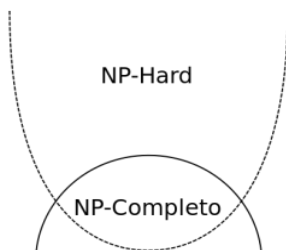


Diagrama de Euler para o conjunto de problemas P, NP, NP-completo, e NP-hard. Imagem da Wikipédia .

**Classe P(Polynomial):** Na classe P, estão os algoritmos que podem ser resolvidos em tempo polinomial, ou seja, o tempo de pior caso é de  $O(n^k)$ , um exemplo são problemas de decisão.

**Classe NP(Nondeterministic polynomial):** Na classe de NP, estão os problemas que são em tempo polinomial por uma máquina de Turing não-determinística, nessa classe estão problemas de busca e otimização.

**Classe NP-completo:** É um subconjunto de NP e São problemas NP que são "mais difíceis" do que outros problemas NP, ou seja, é possível transformar qualquer outro problema NP em um problema NP-completo em tempo polinomial.

**Classe NP-difícil:** São problemas NP para os quais não existe um algoritmo conhecido que possa resolvê-los em tempo razoável. Os problemas NP-

difíceis são tão difíceis quanto os problemas NP-completos.

## **2.3 O Problema do Caixeiro Viajante**

O problema do caixeiro viajante(PCV) é um clássico exemplo de problemas de otimização combinatória. Tendo em vista que pela sua fácil compreensão ele é utilizado, para solucionar vários problemas da sua ampla classe de problemas que são os NP-completo, ele é utilizado em diversas áreas de estudos tal como na utilização de roteamento de veículos, problema de estoque de depósitos, nos raios-x, criptografia, etc.

O problema do caixeiro viajante é um é baseado no ciclo hamiltoniano, no qual é um circuito no grafo hamiltoniano  $G = (V, E)$  o qual ele visita todos os vértices exclusivamente uma vez a fim de encontrar o caminho menos custoso. Deste moto um vendedor que deseja viajar para  $N$  cidades do qual ele só poderá visitar cada uma única vez, nesse caso existiram pela propriedade da permutação haverá  $(N-1)!$  Rotas possíveis sendo uma das cidades o ponto de partida e de retorno. Tanto o problema do caixeiro viajante quanto o problema exposto no trabalho possuem uma relação, ambos buscam visitar todos os pontos mapeados na rota menos custosa. Logo, ambos se utilizam da mesma teoria para elaborar uma solução coerente com a sua realidade.

## **3. Trabalhos Relacionados**

“Otimizacao de Rotas Utilizando Abordagens Heuristicas em Um Ambiente Georreferenciado”. Este artigo discute a aplicação de algoritmos heurísticos na otimização de rotas de entrega de merenda em escolas, incluindo técnicas como Algoritmos Genéticos, Simulated Annealing e Algoritmos de Busca Tabu. Os autores apresentam uma análise comparativa dos métodos e mostram como cada um se aplica a diferentes cenários de entrega. Algumas desvantagem dos métodos heurísticos

apresentados por eles são a não garantia de uma solução ótima, e a instabilidade de suas respostas, pois, podem resultar em soluções diferentes a cada execução. Já o método de força bruta possui a garantia de encontrar a solução ótima, pois, testa todas as possibilidades e garante encontrar a solução ótima.

**"APLICAÇÃO DA METAHEURÍSTICA ALGORITMO GENÉTICO NA OTIMIZAÇÃO DAS ROTAS DE ENTREGAS DA DISTRIBUIÇÃO FÍSICA DE PRODUTOS NO MUNICÍPIO DE FORTALEZA"**

Este artigo aborda a utilização de Algoritmos Genéticos para a otimização de rotas de entrega em uma cidade. Os autores apresentam um estudo de caso com a implementação do algoritmo em uma empresa de entrega e comparam os resultados com outros métodos heurísticos. Porém, apesar de conseguirem um bom resultado em comparação aos algoritmos existentes ainda não possuem a garantia de uma resposta ótima, além da complexidade de implementação do método heurístico utilizado.

## 4. Metodologia

Nesta seção, apresentamos uma maneira de resolver o problema do projeto FlyFood, utilizando um algoritmo de Força Bruta para encontrar o menor caminho ( $\text{argmin}(s(p))$ ). Explicamos de forma didática e clara todo o passo a passo da solução.

### 4.1 Pseudo algoritmo

Classe que representa um local de entrega

```
1: classe House():
2:     metodo_construtor(x, y:Inteiro, name :Caractere):
3:         inicio
4:             parametro.x <- x
5:             parametro.y <- y
6:             parametro.name <- name
7:         fimmetodo
8:     metodo_distant(house :Objeto):
9:         inicio
```

```
10:         retorne valor_absoluto(parametro.x - house.x)
    + valor_absoluto(parametro.y - house.y)
11:     fimmetodo
12: Fimclasse
```

#### Classe para entrada de dados.

```
1:  classe DateInput():
2:      metodo_construtor(url :Caractere)
3:      inicio
4:          parametro.url <- url
5:      fimmetodo
6:      metodo read_file(url:Caractere):
7:      var
8:          line :Vetor
9:      inicio
10:         abrir(parametro.url, 'r') chame obj:
11:         line <- obj.readlines()
12:         retorne line
13:      fimmetodo
14:      metodo create_router(text :Caractere):
15:      var
16:          x, y :Inteiro
17:          position, line, ln :Vetor
18:      inicio
19:          x <- 0
20:          para (line cada item text) faca
21:              ln <- line.replace('\n', '').split(' ')
22:              para (y de 0 ate sizeof(ln) passo 1) faca
23:                  se ln[y] != '0' entao
24:                      position.acrescentar(House(x, y,
ln[y]))
25:              fimse
26:          fimpara
27:          x <- x + 1
28:      fimpara
29:      return position
30:      fimmetodo
31: Fimclasse
```



#### Classe para permutar elementos de um array.

```
1.  função permut(lista :Vetor)
2.  var
3.      lista_aux :Vetor
4.  inicio
5.      se comprimento(lista) <= 1
6.          retorne [lista]
7.      fimse
8.      para i, atual em enumerar(lista)
9.          elementos_restantes = lista[:i] + lista[i+1:]
10.         para p em permut(elementos_restantes)
11.             Lista_aux.extend([atual] + p)
12.         fimpara
13.     fimpara
14.  retorne lista_aux
15. fimfuncao
```

#### Função Intermediária que retorna menor rota

```
1:  funcao (routes :Vetor, r: Objeto)
2:  var
3:      minim, x, y :inteiro
4:      rt :caractere
5:      route, distanci :Vetor
6:  inicio
7:      para (x de 0 ate sizeof(routes) passo 1) faca
8:          route <- ['R']
9:          distanci <- [r.distant(routes[x][0])]
10:         para (y de 0 ate sizeof(routes[x]) passo 1)
11:             route.acrescentar(routes[x][y].name)
12:             se (y < sizeof(routes[x])-1) entao
13:                 distanci.acrescentar(routes[x][y].distant(routes[x][y+1])
14:             )
15:         fimse
16:         fimpara
17:         route.acrescentar('R')
18:         distanci.acrescentar(r.distant(routes[x][-1]))
19:         se (minim = 0 or minim > sum(distanci)) entao
20:             minim <- sum(distanci)
21:             rt <- route
22:         fimse
23:     fimpara
24:  retorne 'A menor rota é %s com exatos %d
25:  dronômetros'%(rt, minim)
26: fimfuncao
```

#### Função para ler o arquivo de entrada

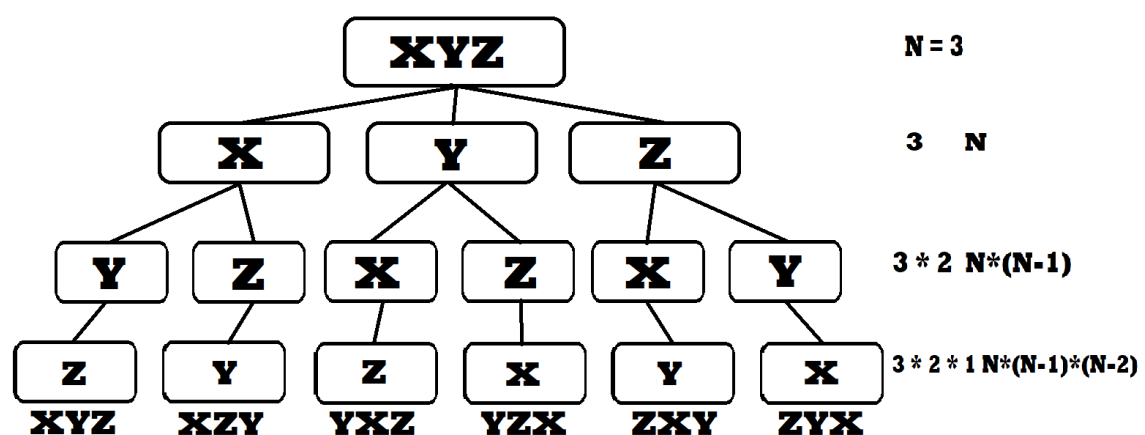
```
1: funcao menor_distancia_file(url)
2:   var
3:     datas, r :Objeto
4:     matriz, routes :Vetor
5:   inicio
6:     datas = DateEntry(url)
7:     matriz = datas.create_matriz(datas.read_file())
8:     routes = datas.create_routes(matriz)
9:     r = list(filtro el: el.name esta em 'R',
10:      routes))[0]
11:     routes = permut(list(filtro el: el.name esta em
12:      'R', routes))
11:     retorne menor(routes, r)
12:   fimfuncao
```

## 4.2 Funcionamento do algoritmo

Para compreendermos melhor a solução que encontramos, primeiramente, precisamos entender o funcionamento de um algoritmo de força bruta. Um algoritmo de Força Bruta é um método para resolver um problema testando todas as possibilidades até encontrar a solução mais adequada. Isso é feito por meio da verificação sistemática de todas as combinações possíveis até que a solução ótima seja encontrada, a qual pode ser encontrada com a utilização de uma permutação dos elementos desejados.

Na função `permut(lista)` é onde está o cerne da solução, ela que executa essa permutação o parâmetro de entrada e um dado do tipo vetor como descrito na 1, na linha 5 até a linha 7 está o caso base da nossa recursão, em seguida temos o início de um laço de repetição na linha 8 do tamanho da lista original, que reservará à primeira posição para o primeiro elemento e assim por diante e criará uma outra lista que será passada adiante para também serem permutados. Visto isso criamos um outro laço de repetição na linha 11 o qual utilizará da mesma função para fazer uma chamada recursiva e gerar todas as permutações, essas permutações serão adicionadas na lista auxiliar e por fim passadas como resultado no fim da função. Essa é a parte com maior complexidade em nosso algoritmo, pois, ela quem faz o algoritmo de força bruta. Criando uma árvore de permutações como vemos abaixo.

A complexidade desse algoritmo é de fatorial, ou seja,  $O(n!)$ . Isso ocorre porque o número de permutações que serão geradas é igual a  $n!$  (fatorial de  $n$ ), onde  $n$  é o comprimento da lista de entrada. Para cada elemento da lista, a função `permut(lista)` é chamada recursivamente, gerando uma nova lista com elementos restantes. Isso significa que para cada elemento da lista original, haverá  $n-1$  elementos restantes, levando a uma quantidade de chamadas recursivas igual a  $(n-1) \times (n-2) \times (n-3) \times \dots \times 2 \times 1$ . O resultado é que a função `permut(lista)` é chamada  $n!$  vezes, o que resulta em uma complexidade de tempo  $O(n!)$ .



A função `menor_dist(url)` é utilizada para calcular a menor distância possível entre uma série de pontos dados. A entrada dessa função é uma URL que aponta para um arquivo de dados contendo informações sobre esses pontos. A ideia principal por trás dessa função é utilizar o algoritmo de permutação para gerar todas as combinações possíveis de trajetos entre os pontos e, em seguida, calcular a distância total de cada uma dessas rotas. A menor dessas distâncias é então retornada como a menor distância possível.

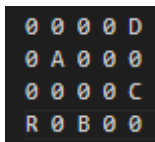
A classe `DateInput` é uma classe que lê dados a partir de um arquivo de texto, ela tem 2 métodos principais: `read_file` e `create_router`. O método `read_file` abre o arquivo na URL especificada e lê as linhas do arquivo, retornando-as como uma lista. O método `create_router()` percorre sobre cada linha na lista de strings e cria um objeto `House` para cada elemento não-nulo (não igual a 0) encontrado. As coordenadas `x` e `y` do objeto `House` são definidas como o índice da linha atual e da coluna atual,

respectivamente. O valor do objeto House é definido como o valor encontrado na lista de strings. A função retorna uma lista de objetos House.

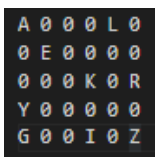
As duas últimas funções "menor\_distancia\_file(url)" e "menor(routes, r)" são funções auxiliares que ajudam a encontrar a menor distância possível entre as casas de uma determinada região. A primeira função "menor\_distancia\_file(url)" usa a classe "DateInput" para ler um arquivo de entrada e criar uma lista de objetos "House", filtrando apenas as casas que possuem uma rota. A segunda função "menor(routes, r)" é responsável por calcular a menor distância entre todas as possíveis rotas, retornando a rota mais curta com a distância total. Essas duas funções são usadas juntas para encontrar a solução final.

## 4. Experimentos

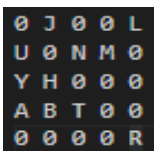
Nos experimentos foram inicialmente testados o caso base do problema, os resultados que foram obtidos a partir de 3 execuções de cada teste, o tempo exibido será em segundos da media de cada testes



Resultado: A menor rota é ['R', 'A', 'D', 'C', 'B', 'R'] com exatos 14 dronômetros, tempo de processamento é instantâneo.



Resultado: A menor rota é ['R', 'L', 'K', 'E', 'A', 'Y', 'G', 'I', 'Z', 'R'] com exatos 22 dronômetros, tempo médio de 0,3906s.



Resultado: A menor rota é ['R', 'L', 'M', 'N', 'J', 'U', 'Y', 'H', 'A', 'B', 'T', 'R'] com exatos 20 dronômetros, tempo médio de 49.8437s.

## 5. Conclusão

Em resumo, o experimento realizado buscou avaliar a eficiência do algoritmo de força bruta para resolver o problema do caixeiro-viajante. O experimento foi realizado em uma máquina com um processador Intel Core i5 de 3.4 GHz, 8 GB de memória RAM e 128 GB de armazenamento em SSD, utilizando o sistema operacional Windows 10 Pro. O experimento foi realizado três vezes e foi utilizado a média dos tempos de execução para obter os resultados.

Os resultados mostraram que o algoritmo de força bruta foi capaz de encontrar a solução do problema do caixeiro-viajante, mas que o tempo de execução aumentou exponencialmente conforme o número de cidades aumentou. Para 4 cidades, o tempo de execução foi Instantâneo, enquanto para 11 cidades foi de 740.8750s. Esses resultados são consistentes com a complexidade do algoritmo, que é em ordem fatorial. Em conclusão, o algoritmo de força bruta mostrou ser uma opção viável para problemas com um pequeno número de cidades, mas é ineficiente para problemas com um grande número de cidades. Dessa forma, futuros trabalhos devem explorar outras alternativas de algoritmos mais eficientes para resolver o problema do caixeiro-viajante.

# Referências Bibliográficas

RODRIGUES, R. A. N.; CARVALHAES, M. F. A. Análise e Complexidade de Algoritmos: Backtracking e Força Bruta. **Revista Ada Lovelace**, [S. l.], v. 1, p. 45–48, 2017. Disponível em: <http://anais.unievangelica.edu.br/index.php/adalovelace/article/view/4117>. Acesso em: 10 jan. 2023.

SILVEIRA, J.f. Porto da. Problema do Caixeiro Viajante. 2000. Disponível em: <http://www.mat.ufrgs.br/~portosil/caixeiro.html>.

Godoy Dotta, Alexandre (2021): USO DE PROGRAMAÇÃO EM ‘R’ APLICADO AO PROBLEMA CAIXEIRO VIAJANTE COM OFICINA PARA DEMOSTRAR DEMAIS APLICAÇÕES. figshare. Presentation.  
<https://doi.org/10.6084/m9.figshare.13857299.v1>

TOVÁ, A. C.; MENDES, O. L. OTIMIZAÇÃO DE ROTAS PARA ENTREGA DOS INGREDIENTES DA MERENDA EM ESCOLAS DA CIDADE DE BARRETOS – SP. Revista Interface Tecnológica, [S. l.], v. 17, n. 2, p. 825–837, 2020. DOI: 10.31510/inf.v17i2.1028. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/1028>

BARBOSA, R. C. Aplicação da metaheurística algoritmo genético na otimização das rotas de entregas da distribuição física de produtos no município de Fortaleza. 90 f. 2014. Dissertação (Mestrado em Logística e Pesquisa Operacional) – Pró-Reitoria de Pesquisa e Pós-Graduação, Universidade Federal do Ceará, Fortaleza, 2014. Disponível em: <https://repositorio.ufc.br/handle/riufc/15038>