Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Bacharelado em Sistemas de Informação

Delivery e algoritmos de menor rota

Joel Fausto de Oliveira Filho

Recife

Agosto de 2022

1. Introdução

O serviço de delivery é algo comum em nossas vidas e tem de muitas formas beneficiado a sociedade com sua praticidade. Este trabalho trata-se da criação de um algoritmo para ajudar a resolução de um problema que possa vir a ocorrer em um futuro não tão distante, que seria o delivery em meio a um trânsito caótico. Visando que nesse futuro o modelo de delivery atual não seria eficiente por questões de tempo e mão-de-obra. Devido a este problema os drones seriam o meio mais eficiente de delivery pois são automáticos e realizam as entregas mais rapidamente, porque não "enfrentam" o trânsito caótico. Porém considerando que a capacidade da bateria dos drones continuam sendo ruim, deve-se obter a rota com maior otimização possível de trajeto, ou seja, deve-se implementar um algoritmo que possibilite aos drones a rota mais otimizada possível.

Serão abordados dois algoritmos para resolução do problema de rota mais otimizada, um algoritmo de "força bruta" e um algoritmo do tipo genético ou do "vizinho mais próximo". Após a elaboração de ambos faremos uma comparação para observar qual é o melhor em aspectos de complexidade para ser implementado.

A motivação do trabalho baseia-se em fornecer um algoritmo com uma solução para o problema em questão (delivery em meio a um trânsito caótico) que possa surgir na sociedade e com essa solução evitar transtornos e estresse para a sociedade de modo geral.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral do trabalho é a criação de um algoritmo com a rota mais otimizada possível, ou seja, com a rota que possua menor distância.

1.1.2 Objetivos Específicos

1° Extrair da matriz os pontos de entrega do drone a partir de um arquivo .txt;

2° Analisar os dados da matriz, a fim de encontrar os pontos que interessam para o cálculo da melhor rota;

3° Fazer permutações com os pontos encontrados, para achar todas as rotas possíveis;

4° Calcular as distâncias entre os pontos;

5° Com as distâncias entre cada ponto, calcular as distâncias de cada rota nas permutações;

6° Examinar dentre todas as distâncias calculadas à menor, pois essa será a rota mais otimizada possível.

2. Referencial Teórico

2.1 Complexidade computacional

Complexidade computacional é a forma encontrada na computação para analisar a eficiência de um algoritmo. Um algoritmo eficiente é aquele que resolve um problema em menor tempo, e dentro da computação é muito importante saber em quanto tempo um algoritmo irá produzir uma resposta, pois existem algoritmos que podem variar de um segundo, até mesmo anos para produzir uma determinada resposta. A complexidade é divida em complexidade espacial (*complexidade espacial refere-se ao espaço de memória que este necessita para executar até o fim*) e complexidade temporal.

Definição 1.1 Segundo Rosen, KH (2009, p.193) "*A complexidade temporal é descrita em termos do número de operações necessárias em vez do tempo atual do computador, por causa da diferença de tempo necessária para computadores diferentes realizarem as operações básicas.*"

2.1.1 Escalas de complexidade

· **Complexidade de melhor caso:** *"Representado pela letra grega Ω (Ômega), é o menor tempo de execução em uma entrada de tamanho N. É pouco usado, por ter aplicação em poucos casos."*

· **Complexidade de caso médio:** *"Definido pela letra grega θ (Theta), é o mais difícil de se determinar. Pois deve-se obter a média dos tempos de execução de todas as entradas de tamanho 1, 2,... até N, ou baseado em probabilidade de determinada situação ocorrer."*

· **Complexidade de pior caso:** *"Representado pela letra grega O (O maiúsculo. Trata-se da letra grega ômicron maiúscula), é o método mais fácil de se obter. Baseia-se no maior tempo de execução sobre as entradas de tamanho N."*

2.1.2 Tipos de complexidade

A complexidade computacional, também possui dois tipos: polinomial e exponencial.

Definição 1.2 *Segundo Rosen, KH (2009, p.197) "Um algoritmo tem complexidade polinomial se tiver complexidade $O(n^b)$, em que b é um número inteiro com $b \geq 1$."*

Definição 1.3 *Segundo Rosen, KH (2009, p.197) "Um algoritmo tem complexidade exponencial se tiver uma complexidade temporal $O(b^n)$, em que $b > 1$."*

2.2 O problema do caixeiro viajante

"No problema do caixeiro-viajante, que está intimamente relacionado ao problema do ciclo hamiltoniano, um vendedor deve visitar n cidades. Modelando o problema como um grafo completo com n vértices, podemos dizer que o vendedor deseja fazer um percurso, ou um ciclo hamiltoniano, visitando cada cidade exatamente uma vez e terminando na cidade de onde partiu. O vendedor incorre em um custo inteiro não negativo c(i, j) para viajar da cidade i para a cidade j, e deseja fazer o percurso cujo custo total seja mínimo, em que o custo total é a soma dos custos individuais ao longo das arestas do percurso." (Cormen, Thomas H. et. al. Algoritmos: Teoria e Prática. Editora Campus, 2002.)

Definição 1.4 *Segundo Rosen, KH (2009, p.656) "Um caminho simples em um grafo G que passe por todos os vértices exatamente uma vez é chamado de caminho hamiltoniano e um ciclo simples em um grafo G que passe pelos vértices exatamente uma vez é chamado de ciclo hamiltoniano."*

2.2.1 Algoritmo de força bruta e o problema do caixeiro viajante

Definição 1.5 *"Na ciência da computação, a busca por força bruta ou busca exaustiva também conhecido como gerar e testar, é uma técnica de solução de problemas trivial, porém muito geral que consiste em enumerar todos os possíveis candidatos da solução e checar cada candidato para saber se ele satisfaz o enunciado do problema."*

Aplicando esta técnica ao problema do caixeiro viajante, testamos todos os caminhos hamiltonianos em busca da melhor rota. Computacionalmente este método para poucos pontos é rápido, porém conforme o número de pontos aumenta, os possíveis caminhos aumentam exponencialmente, fazendo deste método um método com complexidade exponencial como já foi descrito acima, sendo assim este método não é executável para valores "n" muito grandes.

Definição 1.6 "*Para o caso de n cidades, como a primeira é fixa, o leitor não terá nenhuma dificuldade em ver que o número total de escolhas que podemos fazer é (n-1) x (n-2) x ... x 2 x 1. De modo que, usando a notação de fatorial:* ***R(n) = (n - 1)!****.*"

Figura 1 - Possibilidades de rotas em crescimento exponencial.

n	rotas por segundo	(n - 1)!
5	250 milhoes	24
10	110 milhoes	362 880
15	71 milhoes	87 bilhoes
20	53 milhoes	1.2×10^{17}
25	42 milhoes	6.2×10^{23}

3. Procedimento

Para resolução do problema (obter rota com maior otimização possível de trajeto), foram implementados dois tipos de algoritmos, o algoritmo de força bruta, explicado na seção anterior, e o algoritmo do vizinho mais próximo.

3.1 Algoritmo do vizinho mais próximo

Após a aplicação do algoritmo de força bruta para resolução do problema, notou-se que o mesmo possuía uma complexidade muito grande, devido a isso uma nova meta-heurística (método heurístico para resolver problemas de otimização), fora escolhido para a solução do problema, o algoritmo do vizinho mais próximo. A escolha desta meta-heurística tem como objetivo desenvolver um método mais eficiente, ou seja, com menor complexidade ao problema do caixeiro viajante, utilizando uma solução adaptativa.

Definição 1.6 *"O algoritmo do vizinho mais próximo foi, na ciência da computação, um dos primeiros algoritmos utilizados para determinar uma solução para o problema do caixeiro viajante. Ele gera rapidamente um caminho curto, mas geralmente não o ideal."*.

Figura 2 - Função do algoritmo do vizinho mais próximo.

```
função A_Vizinho(lista, dic):

    começo

    rota <- ('R')
    soma <- 0
    ponto <- None

    Para p de 0 até comprimento(lista) Faça
            c, menor <- 0
            Enquanto c < comprimento(lista) Faça
                    d <- calc_distancia(dic[lista[0]][0], dic[lista[c + 1]][0], dic[lista[0]][1],
                                                dic[lista[c + 1]][1])
                    c <- c + 1
                    Se d < menor ou menor == 0 Faça
                            menor <- d
                            ponto <- pontos[c]
            soma <- soma + menor
            lista.remova(rota[-1])
            lista.remova(ponto)
            lista.insere(0, ponto)
            rota.adiciona(ponto)

            // Quando já tiver percorrido todos os pontos, ele deve voltar ao ponto 'R':
            lista.adiciona('R')
            d = calc_distancia(dic[lista[0]][0], dic[lista[1]][0], dic[lista[0]][1], dic[lista[1]][1])
            soma <- soma + d
            rota.adiciona('R')
            rota <- ' → '.junte(rota)

 Retorne f'O menor percurso encontrado foi de {soma} dronômetros, com a rota: {rota}'

    Fim
```

Para implementação do algoritmo do vizinho mais próximo, foi necessário a coleta de dados, obtidos de uma matriz advinda de um arquivo .txt. Os dados obtidos foram introduzidos a uma lista chamada "matriz". De dentro da lista "matriz" foram coletados os pontos de interesse que foram colocados em uma lista chamada "pontos", e em um dicionário chamado "dic_F" foram armazenados como "key" o ponto e como "item" desta "key" o par ordenado ao qual se localiza o ponto dentro da matriz, também foram coletados, os número de linhas e colunas que a matriz possuía. Para o cálculo das distâncias foi criado uma função chamada "calc_distancias", que realiza o cálculo entre dois pares ordenados, tendo como parâmetros respectivamente: x1, y1, x2 e y2, e retorna "distancia" que é a distância entre os dois pares ordenados. E para obtenção do menor percurso e sua respectiva rota, foi usada a lógica do algoritmo do vizinho mais próximo, implementado na função "A_Vizinho" com os parâmetros: lista e dic, que retorna uma string informando a o menor trajeto em dronômetros e a menor rota, como mostrado na Figura 2.

4. Resultados

Figura 3 - Tabela com resultado das comparações entre algoritmos.

DADOS		Tipos de Algoritmos					
		Força Bruta			Vizinho mais próximo		
Matrizes	Qtd. Pontos	Tempo	Rota Encontrada	Distância	Tempo	Rota Encontrada	Distância
MatrizA	3	0.001s	A C D	6	0.001s	A C D	6
MatrizB	5	0.002s	A D C E B	18	0.001s	B A D C E	20
MatrizC	7	0.056s	A D C E G F B	22	0.01s	B A D C E G F	22
MatrizD	10	42.355s	H A D G C E J I F B	32	0.0s	B A H D C G E J I F	32
MatrizE	11	596.576s	H A D G C E K J I F B	34	0.0s	B A H D C G E K I F J	42

A tabela da figura 3, mostra a comparação entre os algoritmos de Força Bruta e Vizinho mais próximo, após uma série de testes de tempo de execução. Em dados, temos o nome da matriz que está sendo avaliada e a quantidade de pontos que a mesma possui. Na parte "Tipos de Algoritmos", vemos os algoritmos que estão sendo comparados, neste caso os usados para resolução do problema (rota mais otimizada), logo abaixo podemos observar o tempo gasto por cada algoritmo em segundos, a melhor rota encontrada e a distância total (ida e volta) que ele obteve.

5. Conclusão

Ao relembrarmos o objetivo principal do trabalho que era, a elaboração de um algoritmo que retornasse a rota mais otimizada possível, ou seja, a rota que com menor distância de ida e volta para determinados pontos, podemos perceber que ele foi alcançado com êxito.

Após a elaboração dos dois algoritmos propostos (Força Bruta e Vizinho mais próximo), obtivemos resultados satisfatórios. Em ambos algoritmos foram retornadas distâncias e rotas que são consideradas “boas”. Porém notou-se uma grande diferença no tempo de execução dos algoritmos, como pode-se notar na tabela da figura 3, localizada nos Resultados. O que nos faz afirmar que em questão de tempo de execução, um algoritmo é melhor que o outro, neste caso o algoritmo do Vizinho mais próximo que nos retornou distâncias “ok” em um tempo bom, quando comparado ao outro algoritmo, que apesar de fornecer distâncias e rotas em alguns casos melhores, que o algoritmo do Vizinho mais próximo possui um tempo de execução ruim. O que ao trazermos para o cenário real é um grande problema, já que por vivermos na era da informação, necessitamos de serviços cada vez mais rápidos para suprir nossas necessidades.

Contudo, pode-se ainda testar diversos outros tipos de algoritmos para execução do problema em questão, visando o desfecho do objetivo principal, ou seja, para trabalhos futuros buscar-se-á a otimização dos algoritmos já experimentados, aspirando uma execução e resultados ainda melhores que os já obtidos ou ainda a implementação de novos tipos de algoritmos, como por exemplo: o algoritmo da colônia de formigas, o algoritmo genético e etc, buscando obter novos resultados que sejam ainda melhores que os obtidos, para um desfecho ainda melhor para o objetivo principal.

Referências Bibliográficas

Rosen, K. H. **Matemática Discreta e suas Aplicações**. Mc Graw Hill: Grupo A, 2010. 9788563308399. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788563308399/. Acesso em: 08 Aug 2022

Cormen, T. **Algoritmos - Teoria e Prática**. GEN LTC: Grupo GEN, 2012. 9788595158092. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595158092/. Acesso em: 08 Aug 2022

Busca por força bruta. **Wikipedia**, 2017. Disponível em: https://pt.wikipedia.org/wiki/Busca_por_for%C3%A7a_bruta#:~:text=Em%20ci%C3%AAncia%20da%20computa%C3%A7%C3%A3o%2C%20busca,satisfaz%20o%20enunciado%20do%20problema. Acesso em: 08 Ago. 2022.

O Problema do Caixeiro Viajante - UFRGS. **UFRGS**, 2000. Disponível em: http://www.mat.ufrgs.br/~portosil/caixeiro.html. Acesso em: 08 Ago. 2022.

Algoritmo do vizinho mais próximo. **Wikipedia**, 2022. Disponível em: https://pt.wikipedia.org/wiki/Algoritmo_do_vizinho_mais_pr%C3%B3ximo#:~:text=O%20algoritmo%20do%20vizinho%20mais,mas%20geralmente%20n%C3%A3o%20o%20ideal. Acesso em: 30 Ago. 2022.