

Otimização do processo de picking em um armazém logístico utilizando algoritmo genético

Danilo Duarte de Jesus Peixoto

Centro Universitário Senac - Santo Amaro

Av. Eng. Eusébio Stevaux, 823 - Santo Amaro, São Paulo

danilo.duartejp@outlook.com

Alessandro Ranulfo Lima Nery

Centro Universitário Senac - Santo Amaro

Av. Eng. Eusébio Stevaux, 823 - Santo Amaro, São Paulo

limanery@yahoo.com.br

RESUMO

A otimização dos processos logísticos dentro de uma empresa da área do varejo é de grande importância, processos claros e otimizados tornará o armazém mais produtivo.

Em vista disso, este trabalho propõe o desenvolvimento de um algoritmo genético na linguagem de programação python, para otimizar um processo logístico chamado *picking* (separação de pedidos), comparando os resultados do algoritmo com uma estratégia de roteirização de forma decrescente com base nos endereços de armazenagem dos produtos. Os resultados obtidos mostraram que o algoritmo é eficiente para otimizar a rota de *picking*, porém para grandes entradas o tempo de execução é relativamente alto, portanto se utilizado em um cenário real, será necessário uma análise do tamanho de lote ideal, considerando a taxa de entrada de pedidos do armazém, para que a geração da rota não atrapalhe o desempenho dos *pickers*.

PALAVRAS CHAVE. Algoritmos genéticos, Roteirização de picking, Gerenciamento de armazém.

ABSTRACT

The optimization of logistical processes within a company in the retail area is of great importance, clear and optimized processes will make the warehouse more productive.

In view of this, this work proposes the development of a genetic algorithm in the python programming language, to optimize a logistical process called picking, comparing the results of the algorithm with a routing strategy in descending order based on the storage addresses of the products. The results obtained showed that the algorithm is efficient to optimize the picking route, but for large entries the execution time is relatively high, so if used in a real scenario, an analysis of the ideal lot size will be necessary, considering the rate of entry of orders from the warehouse, so that the generation of the route does not affect the performance of the pickers.

KEYWORDS. Genetic algorithms. Picking routing. Warehouse management.

1. Introdução

Processos claros e otimizados são fundamentais para alcançar a excelência operacional, o que consequentemente tornará as operações de um armazém mais produtivas e organizadas. No atual cenário das empresas do setor varejista, especialmente aquelas envolvidas no segmento de *e-commerce*, onde é possível comprar praticamente qualquer coisa com apenas alguns cliques do mouse, a entrega se torna um dos aspectos mais importantes. Isso se deve ao fato de que, ao fazer uma compra online, o cliente não tem o produto imediatamente em mãos, resultando em uma experiência de compra diferente daquela em uma loja física Benitez [2021]. O processo de *picking* está diretamente ligado à entrega, pois qualquer atraso nessa etapa pode fazer com que o pedido não entre no próximo horário de coleta da transportadora, atrasando assim a entrega.

Visto que a entrega dentro do prazo é um dos maiores desafios em um *e-commerce* Cavicchioli [2021], como entregar uma boa experiência de compra? Entregando os produtos dentro do prazo, em um mercado que está em constante crescimento. De acordo com um levantamento feito pela Worldpay from FIS, empresa com foco em produtos e serviços financeiros, o mercado do *e-commerce* no Brasil poderá ter um crescimento anual de aproximadamente 18% até 2025 Juan Pablo D'Antiochia [2022]. Assim, otimizando o tempo dos processos realizados dentro de um armazém, poderá agilizar a entrega para o consumidor final. O gerenciamento de armazém é um dos processos logísticos com os quais uma empresa varejista deve se preocupar, já que tudo relacionado ao estoque traz impacto na entrega para o consumidor final, seja no tempo de entrega ou até mesmo na qualidade do produto. Entre as várias operações de um armazém, a separação de pedidos é geralmente a operação mais demorada Roodbergen e De Koster [2001]. Por esse motivo, a separação de pedidos tem prioridade para a melhoria da produtividade De Koster et al. [2007].

O objetivo deste trabalho é reduzir o tempo do processo de *picking* (separação de pedidos) em um layout de estoque baseado em um estoque real de uma empresa no segmento de *e-commerce*, com o uso de um algoritmo heurístico. A redução deste tempo será por meio da otimização da rota do *picker*, priorizando minimizar a distância percorrida, tornando assim este processo mais rápido e agilizando o tempo de entrega para o consumidor final. É possível notar que este problema se trata de um problema de otimização, mais especificamente o problema do caixeiro viajante Hoffman et al. [2013] problema pertencente a classe NP-Completo, este que não possui uma solução exata em tempo polinomial se o problema for grande, tentamos chegar o mais próximo dela possível por algoritmos heurísticos, por esse motivo será desenvolvido um algoritmo genético, que sua origem foi inspirada na teoria da evolução natural de Charles Darwin, Goldberg [1988].

Este problema de otimização de *picking* não é estritamente solucionável por algoritmos heurísticos, como foi no caso do trabalho *An algorithm for dynamic order-picking in warehouse operations* por Wenrong Lu, Duncan Mcfarlane, Vaggelis Giannikas, e Quan Zhang (2015). Neste trabalho foi desenvolvido um algoritmo com base no algoritmo de Ratliff e Rosenthal (1983), que retorna uma solução ótima em tempo polinomial. Entretanto, isso só é possível porque o layout do estoque utilizado no trabalho possui menos de três blocos (zonas) Lu et al. [2015].

Nas próximas seções, serão apresentadas todas as etapas de desenvolvimento do trabalho: a modelagem das restrições, transformando o layout do armazém em um grafo não direcionado, o modelo matemático e, por fim, o desenvolvimento do algoritmo genético para minimizar a distância da rota de *picking* em comparação com uma rota gerada sem um algoritmo.

2. Elaboração do modelo de grafo

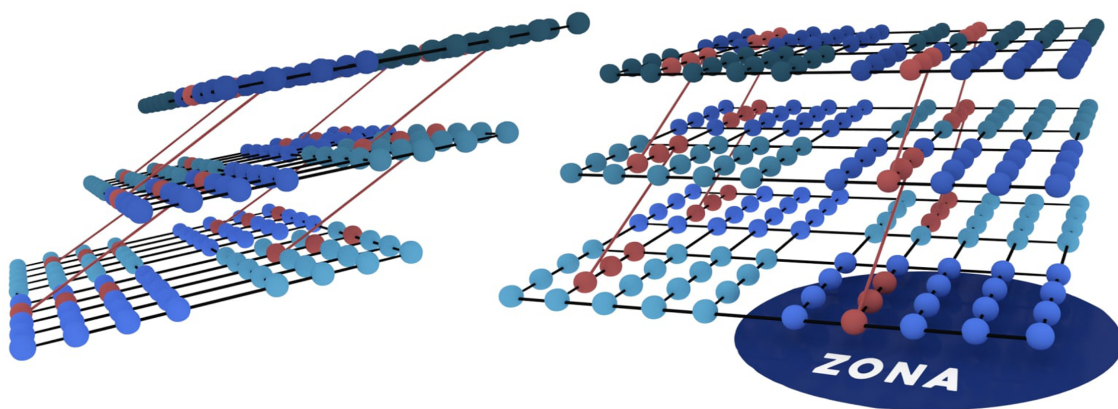
O armazém abordado neste trabalho possui 3 andares, 24 zonas e 4 escadas por andar, e foi desenvolvido utilizando o conceito de grafos. Em um corredor dentro de um armazém real, existem estantes dos dois lados do corredor. Portanto, neste modelo, cada dupla de estantes (lado

direito e lado esquerdo) será representada por apenas um vértice, visto que um picker, estando em uma posição dentro do corredor, é indiferente em pegar o produto na estante à sua direita ou à sua esquerda, devido ao corredor ser estreito. Cada um desses vértices está ligado a outras estantes por arestas, formando os corredores, zonas e andares. As ligações entre andares ocorrem por meio de vértices que representam as escadas. Todas essas ligações possuem um peso, que é a distância entre dois vértices. Neste modelo, existem 4 valores de pesos: peso 1 para arestas dentro do mesmo corredor, peso 2 para arestas que ligam um corredor a outro, peso 3 para arestas entre zonas e peso 4 para arestas entre andares. Este modelo foi desenvolvido utilizando queries SQL e a biblioteca iGraph da linguagem Python, uma biblioteca open source utilizada para manipulação e análise de grafos. Primeiramente, foram criadas tabelas em um banco de dados SQL Server para armazenar os locais de estoque, coordenadas desses locais, vértices representando cada local, as conexões entre os vértices com base nas coordenadas e uma tabela representando a fila de lotes aguardando o picking.

As conexões entre os vértices foram realizadas por meio de consultas SQL, conectando locais adjacentes e os andares. É possível verificar se um local é adjacente a outro por meio de seu endereço armazenado na tabela "locais_estoque". Esses endereços seguem um padrão. Por exemplo, um endereço pode ser: 2.A1.06.16.03.02. O endereço possui 6 informações: o primeiro é o andar (2), o segundo é a zona (A1), o terceiro é o corredor (06), o quarto é a estante dentro do corredor (16), o quinto é o nível na estante (03) e, por fim, um número que varia de 1 a 2, representando o bin (02), que são caixas. Nesse caso, o produto estaria na segunda caixa. Após a criação de todas as conexões, o código Python consome a tabela "local_conexoes", passando os vértices 1 e 2 para uma função da biblioteca iGraph, que cria o grafo a ser utilizado no algoritmo.

A figura 1 é um exemplo reduzido do grafo do armazém, com apenas 12 zonas e uma quantidade reduzida de estantes por corredor. Os vértices e arestas em vermelho representam as escadas e suas conexões.

Figura 1: Grafo Armazém



3. Elaboração do modelo Matemático

O modelo matemático do problema de roteirização de picking, será basicamente o mesmo modelo do problema do caixeiro viajante, que pode ser formulado como um programa linear inteiro, neste caso será utilizado a formulação de Miller et al. [1960].

Definição das variáveis:

- X_{ij} , variável de decisão binária, 1 se o vértice j está ligado com o vértice i por uma aresta e 0 caso contrário.

- C_{ij} , peso da aresta entre o vértice i e o vértice j .
- u_i , número de vértices visitados incluindo o vértice i .
- n , número de vértices.

$$X_{ij} = \begin{cases} 1, & \text{se a rota incluir uma ligação direta entre os locais } i \text{ e } j \\ 0, & \text{caso contrário.} \end{cases}$$

$i, j \in \{1, 2, \dots, n\}$

Função objetivo:

$$\min \sum_{i=1}^n \sum_{j=1, j \neq i}^n C_{ij} X_{ij} \quad (1)$$

Restrições: Restrição de entrada. Cada local deve ser visitado a partir de um único local:

$$\sum_{i=1, i \neq j}^n X_{ij} = 1, j \in \{1, 2, \dots, n\} \quad (2)$$

Restrição de saída. A partir de um local é possível visitar apenas um próximo local:

$$\sum_{j=1, j \neq i}^n X_{ij} = 1, i \in \{1, 2, \dots, n\} \quad (3)$$

Restrições de subrotas:

$$\begin{aligned} u_i - u_j + nx_{ij} &\leq n - 1, \quad i, j \in \{1, 2, \dots, n\}, i \neq j, \\ 2 &\leq u_i \leq n \quad i \in \{2, \dots, n\} \\ u_i &\in \mathbb{R}^+, i \in \{2, \dots, n\}, u_1 = 1 \end{aligned} \quad (4)$$

4. Implementação do algoritmo genético para roteirização

Após a modelagem do armazém em forma de grafo, foi desenvolvido um algoritmo genético para otimizar a rota de picking. Antes da execução do algoritmo, é verificado se existe um arquivo no projeto contendo todas as distâncias do grafo entre os vértices i e j , para todos os possíveis pares de vértices. Caso o arquivo não exista, é acionado o algoritmo desenvolvido para calcular essas distâncias e armazená-las em um arquivo CSV. Se o arquivo já existe, as distâncias são armazenadas em uma estrutura de dicionário em Python, para posterior consumo pelo algoritmo. Em seguida, é realizada a busca pelos lotes de separação no banco de dados, retornando todos os produtos presentes em cada lote, juntamente com seus respectivos endereços.

Antes da execução do método principal, é enviado a lista de lotes para um método que analisa endereços duplicados. Isso ocorre porque, para o algoritmo de roteirização, é importante considerar o endereço apenas até a informação da prateleira. Portanto, se houver dois produtos em um lote na mesma prateleira, mas em níveis diferentes, ambos os produtos seriam associados ao mesmo identificador de vértice, o que geraria duplicidade. Nesses casos, o analisador de duplicidade gera um novo vértice auxiliar para o endereço duplicado, copiando as conexões do vértice original, de modo que os dois produtos sejam representados por vértices diferentes.

Terminado todas as verificações e tendo todos os dados necessários para o funcionamento do algoritmo genético, é realizada sua execução, recebendo 5 parâmetros, sendo eles:

- O grafo do armazém
- A rota inicial sendo a rota gerada sem algoritmo, apenas ordenando a busca pelos endereços de forma decrescente
- O número identificador do lote
- Dicionário com as distâncias do arquivo csv
- O número de indivíduos que serão escolhidos para o crossover

O algoritmo é iniciado gerando uma população inicial aleatória. O número de indivíduos nessa população é igual a 1 somado ao número máximo de combinações dois a dois, com base no número de pais que serão selecionados na etapa de seleção. Esse "1" adicional representa a rota inicial, que é necessária na população para evitar o risco de o algoritmo retornar uma solução pior do que a rota inicial. Cada indivíduo na população é representado por um array, onde cada índice do array representa um gene, ou seja, um vértice que representa um local onde os produtos do lote estão armazenados.

Após a geração da população inicial, o algoritmo entra em um laço de repetição que executa as operações referentes às gerações. O critério de parada para a obtenção da melhor solução é se passarem 200 gerações sem nenhuma evolução nos indivíduos. Nesse caso, o algoritmo sai do laço e retorna a melhor solução encontrada até então. Ao longo das gerações (repetições no laço), são realizadas as operações de seleção, *crossover* e mutação.

A primeira operação realizada é a seleção, utilizando o método de seleção por truncamento. Esse método seleciona os n indivíduos mais aptos da população atual, onde n é o número de indivíduos escolhidos para o *crossover*, conforme definido no início do algoritmo. A escolha dos indivíduos mais aptos é feita com base no seu *fitness*, também conhecido como valor de aptidão. Esse valor é calculado no método *fitness* desenvolvido, o cálculo do *fitness* é dado pelo seguinte somatório:

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^n D_{ij} \quad (5)$$

Assim, a distância total da rota é obtida somando todas as distâncias do vértice i para o vértice $i+1$, percorrendo todo o vetor de genes do indivíduo. Esse valor total de distância é então retornado como o valor de *fitness* do indivíduo, multiplicado por -1, com o objetivo de minimizar a distância e maximizar o *fitness*. Vale ressaltar que o cálculo da distância do vértice i para o vértice $i+1$ não é necessário, uma vez que essas distâncias já foram calculadas previamente e armazenadas em um arquivo CSV, que foi convertido em um dicionário. Dessa forma, para encontrar a distância entre o vértice i e o vértice $i+1$, basta buscar no dicionário pela chave formada pela concatenação desses dois vértices.

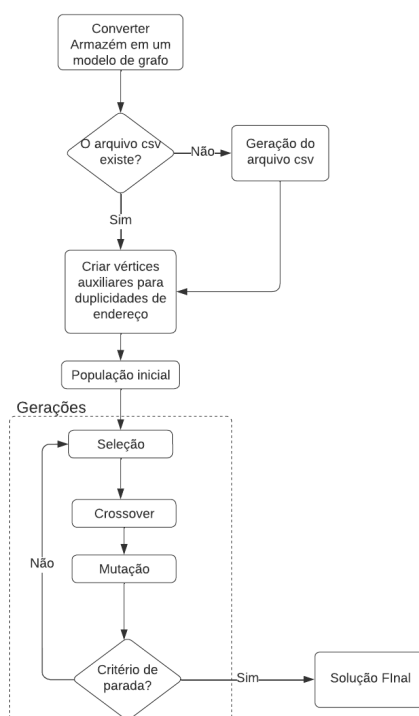
Em seguida à operação de seleção, é realizada a operação de *crossover* nos indivíduos selecionados, comumente referidos como "pais". No entanto, antes que qualquer operação seja executada, é garantido que n pais não participem do *crossover*, a fim de preservar a diversidade da população. O valor de n precisa ser fornecido previamente no algoritmo. No *crossover*, é escolhido um ponto de corte aleatório, x , que varia de 0 ao tamanho do indivíduo. Esse ponto de corte divide os genes dos pais, gerando duas partes distintas: cabeças e caudas. As caudas dos pais são então trocadas, resultando em dois novos indivíduos filhos. Essa operação de *crossover* é realizada em todas as combinações possíveis entre os indivíduos pais, gerando uma nova população. Por esse motivo, o

tamanho da população inicial é definido como 1 mais o número máximo de combinações possíveis entre os pais, garantindo que a população permaneça com o mesmo tamanho após o *crossover*.

A última operação realizada em uma geração é a mutação. Diferente do *crossover*, onde um número n de indivíduos é selecionado para não ser submetido a alterações, na mutação é escolhida uma taxa de mutação que representa a probabilidade de um indivíduo sofrer mutação. Na etapa de mutação, dois números aleatórios, x e y , são escolhidos com valores entre 0 e o tamanho do indivíduo. Após a seleção dos números, ocorre a troca dos valores dos genes nos índices x e y .

Ao final de todas as operações realizadas na geração, é calculado o *fitness* do indivíduo mais apto até o momento. Esse valor de *fitness* é comparado com o indivíduo mais apto da próxima geração, repetindo esse processo até que 200 gerações se passem sem nenhuma evolução. Nesse momento, é retornado o melhor resultado de rota encontrado até o momento.

Figura 2: Fluxograma Algoritmo roteirização de picking



5. Resultados

O algoritmo foi executado para 300 lotes de separação gerados aleatoriamente, com tamanhos variados. A rota gerada pelo algoritmo foi comparada com a rota gerada sem o uso do algoritmo, na qual os produtos foram apenas ordenados de forma decrescente com base nos endereços de armazenagem. Após a execução de cada lote, os resultados foram armazenados em um arquivo CSV. Para facilitar a análise, os dados dos 300 lotes foram importados para uma tabela em um banco de dados SQL.

Média de ganho (%)	Média de tempo de execução	Ganho máximo (%)	Ganho mínimo (%)
17,2546792604213	17,3201431888545	45,6730769230769	0

Tabela 1: Médias e ganhos máximo e mínimo

A Tabela 1 mostra o resultado de uma consulta SQL nos dados importados do CSV, fornecendo 4 informações importantes para a análise. Em média, o ganho de otimização da rota com o algoritmo de roteirização é de 17,25%. Isso significa que dois *pickers* que estão trabalhando no mesmo lote e cada um utilizando uma forma diferente de ordenação de busca, o que está seguindo a rota proposta pelo algoritmo de roteirização percorrerá, em média, um caminho 17,25% mais curto em comparação com a rota percorrida pelo o outro *picker* utilizando a estratégia decrescente. Quanto ao tempo médio de execução, o algoritmo leva cerca de 17,32 segundos para retornar uma rota para um lote. Esse tempo é proporcional ao tamanho do lote, como pode ser observado no gráfico de dispersão apresentado na Figura 3.

Figura 3: Média de tempo por quantidade de produtos



O ganho máximo obtido para a execução dos 300 lotes foi de 45,67%, enquanto o ganho mínimo foi de 0,0%. Apenas em 1 lote entre os 300 executados, o algoritmo não conseguiu otimizar a rota. Ao analisar os endereços dos produtos nos dois lotes mencionados, o lote com ganho máximo e o lote com ganho mínimo, é possível perceber que os produtos do lote com ganho máximo estão mais dispersos pelas zonas e andares do armazém.

Figura 4: Lotes ganho máximo e mínimo

Lote ganho mínimo			Lote ganho máximo		
LOTE	PRODUTO	ENDEREÇO	LOTE	PRODUTO	ENDEREÇO
178	Produto6	1.A1.03.21.05.02	2	Produto8	1.C2.04.25.05.05
178	Produto5	1.A1.05.07.04.02	2	Produto10	2.A2.01.01.05.01
178	Produto2	2.A1.02.02.06.02	2	Produto5	2.A2.02.05.02.01
178	Produto1	2.A1.07.12.01.04	2	Produto3	2.A2.05.03.01.02
178	Produto3	2.B2.01.04.03.05	2	Produto1	2.B1.02.07.06.02
178	Produto4	2.B2.05.16.02.02	2	Produto4	2.B1.08.03.02.02
			2	Produto6	2.C1.02.06.01.05
			2	Produto2	2.C2.07.03.04.04
			2	Produto9	3.A1.03.01.03.02
			2	Produto7	3.A2.05.26.03.01

Analizando a Figura 4, apesar da diferença de tamanho, é possível observar que o lote 178 possui os produtos posicionados próximos uns dos outros. Os produtos estão distribuídos nos andares 1 e 2, sendo que no andar 1 estão concentrados apenas na zona A1, enquanto no andar 2 estão nas zonas A1 e B2, com dois produtos em cada zona. Por outro lado, no lote 2, os produtos estão dispersos nos três andares e apresentam uma maior diversidade de zonas.

O algoritmo só conseguirá otimizar a rota caso os produtos estejam amplamente distribuídos. Caso os produtos já estejam próximos, a estratégia decrescente baseada nos endereços já apresenta resultados satisfatórios.

6. Conclusão

A roteirização de *picking* utilizando algoritmo genético mostrou-se eficiente na redução da distância percorrida pelo *picker*. Embora nem sempre retorne a solução ideal, em armazéns que não utilizam nenhum algoritmo para geração de rota, apenas estratégias como a ordenação decrescente com base no endereço, é possível obter uma melhora significativa no tempo de *picking*, dependendo do layout e tamanho do armazém. Isso ocorre porque, em armazéns de pequeno porte, os produtos já estão próximos uns dos outros, o que torna a otimização com o algoritmo inviável, como demonstrado na seção 5.

Mesmo que o algoritmo tenha se mostrado eficiente para a roteirização, seu tempo de execução aumenta consideravelmente à medida que a quantidade de produtos no lote aumenta, dependendo do valor da taxa de entrada de pedidos em um armazém e do tamanho dos lotes o tempo necessário para o algoritmo otimizar a rota pode acabar inviabilizando seu uso. Portanto, ao optar por utilizar o algoritmo, é importante ponderar a escolha do tamanho dos lotes, buscando um equilíbrio. Lotes muito pequenos não exigiriam o uso do algoritmo genético, uma vez que seria possível desenvolver um algoritmo determinístico que otimizasse a rota em tempo polinomial.

Referências

- Benitez, J. (2021). "a importância dos prazos de entrega no seu e-commerce". Disponível em: <https://www.ecommercebrasil.com.br/artigos/prazos-de-entrega-no-e-commerce-importancia/>. Acesso em: 06 de junho 2022.
- Cavicchioli, R. (2021). "os cinco maiores desafios para um e-commerce de sucesso". Disponível em: <https://www.ecommercebrasil.com.br/artigos/os-cinco-maiores-desafios-para-um-e-commerce-de-sucesso/>. Acesso em: 06 de junho 2022.
- De Koster, R., Le Duc, T., e Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, p. 481–501.
- Goldberg, D. E. (1988). Genetic algorithms in search optimization and machine learning.
- Hoffman, K. L., Padberg, M., Rinaldi, G., et al. (2013). Traveling salesman problem. *Encyclopedia of operations research and management science*, 1:1573–1578.
- Juan Pablo D'Antiochia (2022). "expansão dos métodos de pagamentos digitais no e-commerce da américa latina". Disponível em: <https://www.ecommercebrasil.com.br/artigos/metodos-de-pagamentos-digitais/>. Acesso em: 06 de junho 2022.
- Lu, W., Mcfarlane, D., Giannikas, V., e Zhang, Q. (2015). An algorithm for dynamic order-picking in warehouse operations. *European Journal of Operational Research*, p. –.
- Miller, C. E., Tucker, A. W., e Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.
- Roodbergen, K. J. e De Koster, R. (2001). Routing order pickers in a warehouse with a middle aisle. *European Journal of Operational Research*, 133:32–43.