DESENVOLVIMENTO DE UM MÉTODO BASEADO NO ALGORITMO DE DIJKSTRA PARA OTIMIZAÇÃO DE ROTAS NO PROCESSO DE COMPRAS DE PRODUTOS

DEVELOPMENT OF A METHOD BASED ON DIJKSTRA'S ALGORITHM FOR ROUTE OPTIMIZATION IN THE PRODUCT PURCHASING PROCESS

Rondinele Florêncio de Oliveira*
Henrique Viana Oliveira**

RESUMO

O algoritmo de Dijkstra é um algoritmo da Teoria dos Grafos que determina o caminho mais curto entre vértices de um grafo ponderado sem ciclos negativos. No estudo apresentado é implementada uma variação deste algoritmo para determinar a melhor maneira de comprar uma lista de produtos em redes de lojas, representadas como um grafo, havendo a possibilidade de diferentes produtos serem comprados em diferentes lojas. Para realizar essa tarefa, são utilizadas duas abordagens diferentes: a primeira abordagem analisa um cenário em que todos os produtos estão disponíveis em todas as lojas e a compra deve ser feita em apenas uma loja; e a segunda abordagem permite comprar diferentes produtos em diferentes lojas e considera que as lojas possuem estoque limitado dos produtos. Como resultado foram implementados dois algoritmos referentes às duas abordagens e realizadas as suas respectivas análises de complexidade.

Palavras-chave: Dijkstra. Algoritmos de Caminho Mínimo. Grafos.

ABSTRACT

Dijkstra's algorithm is a Graph Theory algorithm that determines the shortest path between vertices of a weighted graph without negative cycles. The presented study presents a variation of this algorithm to determine the best way to buy a list of products in chain stores, represented as a graph, with the possibility of different products being purchased in different stores. To accomplish this task, two different approaches are used: the first approach analyzes a scenario in which all products are available in all stores and the purchase must be made in only one store; and the second approach allows buying different products in different stores and considering that the stores have a limited stock of the products. As a result, two algorithms referring to the two approaches were implemented and their respective complexity analyzes were carried out.

Keywords: Dijkstra. Shortest-Path Algorithms. Graph.

---

* Discente do Bacharelado em Ciência da Computação.
** Possui mestrado e doutorado em Ciência da Computação pela Universidade Federal do Ceará(UFC)

1 INTRODUÇÃO

A Teoria dos Grafos é um ramo da matemática que estuda a relação entre objetos abstratos chamados de vértices e arestas. O objetivo principal da Teoria dos Grafos é modelar situações reais por meio de grafos e desenvolver algoritmos que possam ser aplicados a esses modelos para resolver problemas complexos. A partir dela, foram criados diversos algoritmos de busca de caminhos modernos, como busca em largura, busca em profundidade, árvores geradoras mínimas e o problema de caminho mais curto. (CORMEN et al., 2009).

Os algoritmos de caminho mais curto (*shortest path*) (CORMEN et al. 2009, cap 24) consistem na minimização do custo de travessia entre dois vértices de um grafo, custo este dado pela soma dos pesos de cada aresta percorrida. Eles são amplamente utilizados para obtenção de rotas em diversas áreas de atuação, como cálculo de rotas IP (SALES; SANTOS et al., 2019), mapeamento de rotas de entregas (CORREIA, 2019), aplicativos de navegação (SILVA; ARAÚJO et al., 2020), entre outras áreas, e se estendem até em criações de rotas com problemas de caminho em grafos.

Para calcular o caminho mais curto de um grafo, é necessário executar uma busca no grafo, que significa seguir sistematicamente as arestas do grafo de modo a visitar os vértices do grafo (CORMEN et al., 2009). As técnicas para executar buscas em um grafo estão no núcleo da área de algoritmos em grafos e um algoritmo de busca pode revelar muita coisa sobre a estrutura de um grafo. Para destacar a relevância do problema de caminho mais curto, algoritmos da área de Processamento de Rotas em grafos estão cada vez mais sendo requisitados, exemplo (CHEN, 2022), (SIDABUTAR; SINULINGGA, 2022).

Existem dois tipos principais de algoritmos de caminho mais curto: os de caminho único (*single path*) e os de todos os pares de caminhos (*all paths*). Alguns exemplos de algoritmos de caminho único, são algoritmo de Dijkstra ((CORMEN et al., 2009),cap 24.3), e o algoritmo A* (HART; NILSSON; RAPHAEL, 1968) que encontram o caminho mais curto entre um par de vértices em um grafo direcionado ou não-direcionado. Já para os algoritmos de todos os pares de caminhos, temos o exemplo do algoritmo de Floyd-Warshall (FLOYD, 1962), que encontra os caminhos mais curtos entre todos os pares de vértices em um grafo. A principal diferença entre esses algoritmos é a sua aplicação e complexidade. Algoritmos de caminho único são geralmente mais rápidos e eficientes quando se deseja encontrar apenas um caminho mais curto, enquanto que algoritmos de todos os pares de caminhos são mais lentos, porém mais apropriados quando se deseja encontrar todos os caminhos mais curtos em um grafo.

Nesse sentido, vários problemas se enquadram nos quesitos abordados anteriormente. Alguns algoritmos de Teoria dos Grafos têm sido utilizados com sucesso para melhorar questões financeiras. Por exemplo, aplicação de um algoritmo de Dijkstra aprimorado para o planejamento inteligente de rotas de navios (ZHU et al., 2021) que ao minimizar o tempo de viagem e melhorar a utilização dos recursos, é possível aumentar a eficiência operacional e, consequentemente, reduzir os custos associados ao transporte marítimo. Da mesma forma, (BHATTACHARYYA; BANERJEE; KÖPPEN, 2022) propõe a utilização do algoritmo de Dijkstra com técnicas in-

teligentes de gerenciamento de energia para o planejamento de caminhos ótimos e encontrar o caminho mais curto com consumo mínimo de energia, considerando parâmetros de rota e proporcionando uma melhor eficiência veicular e utilização de recursos. Essas aplicações dos algoritmos trouxeram avanços e facilidades para as pessoas, poupando tempo e dinheiro em suas atividades cotidianas.

Neste trabalho, a problemática envolve a necessidade de comprar uma lista de produtos de maneira mais eficiente em uma cidade. Sendo assim, este trabalho implementa um algoritmo que ajuda a situação dessas pessoas, utilizando técnicas de Teoria dos Grafos para tornar a tarefa de compras mais financeiramente viável.

De maneira mais objetiva, este trabalho desenvolve uma variação do algoritmo de *Dijkstra* capaz de descobrir o melhor caminho para comprar uma lista de produtos em várias lojas. O problema permite trabalhar com uma série de lojas (vértices do grafo) interligadas por estradas/vias (as arestas do grafo), onde cada loja contém uma lista de produtos com seus respectivos preços. Além disso, é considerada a lista de compras do consumidor. O primeiro algoritmo proposto busca uma única loja para comprar os produtos, sendo essa a versão simplificada do problema. Já o segundo algoritmo é a generalização do problema, levando em conta a possibilidade de não haver estoque suficiente nas lojas e apresentando um caminho indicando a quantidade e preço dos produtos a serem comprados no percurso. Com essa variação do algoritmo de *Dijkstra*, é possível calcular os caminhos mínimos contextualizando a economia do cliente na compra dos produtos, proporcionando uma economia financeira para o consumidor.

O presente trabalho está estruturado como segue. A Seção 2 apresenta os trabalhos relacionados à abordagem proposta. Na Seção 3 o referencial teórico aborda as tecnologias e conceitos utilizados na construção dos algoritmos, enquanto a Seção 4 descreve a metodologia dos algoritmos criados e utilizados. Os resultados obtidos são apresentados na Seção 5, e a Seção 6 finaliza o artigo com uma síntese dos resultados obtidos neste trabalho.

2 TRABALHOS RELACIONADOS

Existem muitos problemas que podem ser representados e resolvidos utilizando grafos. Por exemplo, mapas podem ser representados em um grafo. Neste caso, os vértices correspondem aos endereços, cidades ou pontos no mapa e as arestas (ou as ligações entre vértices) correspondem às estradas, ruas e pontes conectando locais no mapa. Visto isso, esta seção apresenta artigos relacionados ao tema de caminhos mínimos em grafos. Os artigos citados foram utilizados como base para seleção do algoritmo utilizado neste trabalho. Também levantamos os aspectos relativos ao tema de otimização da compra de produtos e decidimos a melhor forma de abordar a problemática apresentada.

No artigo proposto por SOUZA; MAGALHãES; MACHADO(2018) foi apresentado o desenvolvimento de uma aplicação móvel destinada à otimização de compras em supermercados, permitindo aos usuários identificar o estabelecimento com a melhor oferta de preços. A proposta utiliza o conceito de computação coletiva, que envolve a obtenção de informações sobre preços e

promoções de produtos por meio da colaboração de uma comunidade de consumidores. Além disso, a aplicação utiliza conceitos de pesquisa operacional e algoritmos de programação linear para calcular a melhor compra para o usuário.

Em MARTINS(2015) foi apresentada uma abordagem para otimizar as compras, utilizando algoritmos de caminho mínimo e busca de possibilidades. O estudo concluiu que, a fim de obter a compra mais econômica, é recomendado realizar compras em diferentes estabelecimentos, levando em consideração os preços praticados por cada um. Portanto, a solução proposta consiste em determinar a rota mais eficiente para percorrer os supermercados, permitindo que o consumidor adquira os produtos com as melhores ofertas disponíveis.

Já em PEREIRA; OLIVEIRA(2019) foi desenvolvido um algoritmo para auxiliar os usuários durante a compra de uma lista de produtos visando economia, levando em consideração o preço total de uma lista de produtos e o custo com o deslocamento do até esse(s) supermercado(s). O trabalho desenvolveu um algoritmo que realiza combinações de lojas e produtos a fim de descobrir a melhor forma de comprar a lista de produtos. O estudo concluiu que é viável a criação do algoritmo que retorna a combinação de duas lojas entre as n lojas de entrada.

Abaixo, encontra-se a Tabela 1 de comparação dos artigos relacionados, juntamente com este trabalho, destacando os principais aspectos considerados. A tabela apresenta informações sobre a presença de otimização, o uso de grafos, o tempo de execução no contexto de tempo polinomial, a consideração de múltiplas lojas na resposta e a consideração da quantidade de produtos por loja. Essa comparação permite visualizar de forma concisa as diferenças e semelhanças entre os trabalhos referenciados e o presente estudo, evidenciando as contribuições distintivas deste trabalho em relação ao estado da arte.

Tabela 1 – Tabela de Comparação de Artigos

Artigo	Otimização	Grafo	Polinomial	Múltiplas Lojas	Quantidade
SOUZA; MAGALHãES; MACHADO (2018)	SIM	NÃO	NÃO	NÃO	NÃO
MARTINS(2015)	SIM	SIM	NÃO	NÃO	NÃO
PEREIRA; OLIVEIRA (2019)	SIM	SIM	SIM	SIM	NÃO
Este Trabalho	SIM	SIM	SIM	SIM	SIM

Fonte: Elaborado pelos autores..

O trabalho desenvolvido neste artigo tem como objetivo aprimorar a eficiência na compra de produtos, através da busca da melhor maneira de percorrer um grafo de lojas para adquirir uma lista de produtos. Dessa forma, o algoritmo desenvolvido permite a compra em várias lojas diferentes, otimizando o tempo e o custo dos produtos. Vale ressaltar que, diferentemente dos outros trabalhos citados, esse trabalho irá envolver quantidades diferentes de produtos nas lojas, assim trazendo maior otimização do problema de comprar uma lista de produtos.

3 REFERENCIAL TEÓRICO

Para facilitar o entendimento do algoritmo neste trabalho, esta seção tem como objetivo apresentar os conceitos e notações básicas sobre grafos e seus algoritmos de caminho mínimo. Mais detalhes sobre a Teoria dos Grafos podem ser encontrados em (CORMEN et al., 2009).

Um grafo $G = (V, E)$ é uma estrutura de dados que consiste em um conjunto de vértices V e arestas E. O conjunto V representa um conjunto não vazio de nós do grafo, enquanto o conjunto E representa um conjunto de pares ordenados, em que cada $e \in E$ é dado por $e = (u, v)$, para $u, v \in V$. O peso ou custo de uma aresta de um grafo é definido pela função $w : E \to \mathbb{R}$, que é o peso individual ou custo não negativo associado a uma aresta.

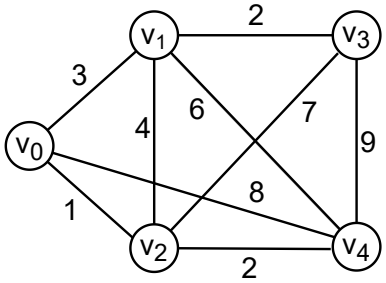
Um caminho é uma lista de vértices sequenciais que conectam um vértice de origem u a um vértice de destino v. Em um problema de caminho mais curto, consideramos um grafo ponderado $G = (V, E)$ com função peso de aresta $w : E \to \mathbb{R}$. Dado o caminho $p = \langle v_1, v_2, v_3, ..., v_k \rangle$, o peso do caminho p é dado por $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.

O peso do menor caminho de u até v, denotado por $\delta(u, v)$, é a soma dos pesos das arestas presentes no caminho de u até v. O peso do menor caminho entre dois vértices u e v pode ser definido como:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{se existe um caminho de u para v,} \\ \infty & \text{caso contrário,} \end{cases}$$

onde $u \overset{p}{\rightsquigarrow} v$ significa um caminho p de u até v. O peso do menor caminho de u até v pode ser visto como o peso do menor caminho de u até v. Considere o seguinte grafo G:

Figura 1 – Exemplo de um grafo.



Fonte: Elaborado pelos autores.

O grafo da Figura 1 apresenta um conjunto de vértices V, tal que $V = \{0, 1, 2, 3, 4\}$, com o conjunto de arestas $E = \{(0,1), (0,2), (0,4), (1,0), (1,2), (1,3), (1,4), (2,0), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4), (4,0), (4,1), (4,2), (4,3)\}$ e peso das arestas $w(0,1) = w(1,0) = 3$, $w(0,2) = w(2,0) = 1$, $w(0,4) = w(4,0) = 8$, $w(1,2) = w(2,1) = 4$, $w(1,3) = w(3,1) = 2$, $w(1,4) = w(4,1) = 6$, $w(2,3) = w(3,2) = 7$, $w(2,4) = w(4,2) = 2$ e $w(3,4) = w(4,3) = 9$. Exemplos de caminhos dos vértices de 0 até 4 podem ser dados por $p_1 = \langle 0,1,4 \rangle$ ou $p_2 = \langle 0,4 \rangle$. Já o menor caminho entre os vértices 0 e 4 é dado por $p = \langle 0,2,4 \rangle$, pois tem o menor custo, ou seja,

$w(p) = 3$. Observe que o menor caminho ou caminho mais curto é definido como o caminho de menor custo, e não o caminho com a menor quantidade de vértices.

Na literatura, para encontrar o caminho mais curto entre vértices, podemos utilizar o algoritmo de Dijkstra, que será explicado na sequência. A algoritmo tem como entrada um grafo G, o peso das arestas w e um vértice de origem s, e como saída produz os caminhos mínimos do vértice de origem para todos os outros vértices do grafo.

A maioria dos algoritmos que funcionam em grafos precisa manter atributos para vértices e/ou arestas. Indicamos esses atributos usando nossa notação usual, por exemplo, *v.dist* para um atributo *dist* de um vértice v. Para o algoritmo de Dijkstra são utilizados dois atributos: *v.dist*, que representa o menor caminho do vértice de origem até o vértice v; e *v.pai*, que representa quem é o vértice antecessor a v no menor caminho do vértice de origem até v.

Algoritmo 1 DIJKSTRA(G, w, s)

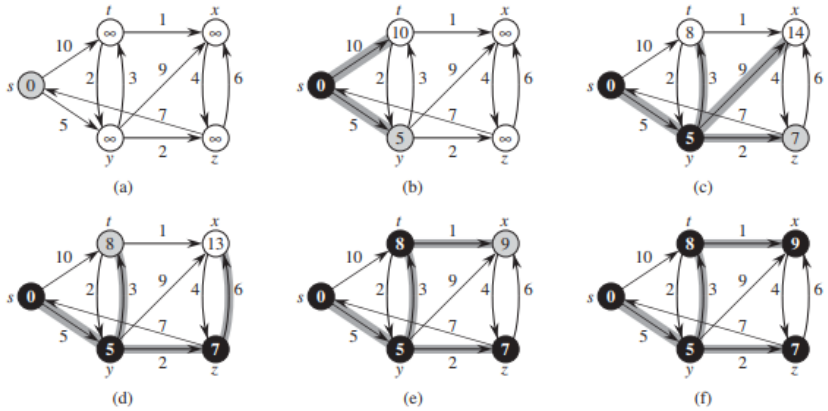
```
1:  para cada vértice v ∈ G.V faça                ▷ Inicialização das distâncias dos vértices
2:      v.dist ← ∞
3:      v.pai ← NIL
4:  s.dist ← 0                                    ▷ Distância do nó de origem s
5:  Q ← G.V                                       ▷ Fila de prioridade com todos os nós do grafo
6:  enquanto Q ≠ ∅ faça
7:      u ← EXTRACT-MIN(Q)                        ▷ Remoção do nó com menor distância de Q
8:      para cada vértice v ∈ G.ADJ(u) faça       ▷ Atualização das distâncias dos nós adjacentes
9:          se v.dist > u.dist + w(u,v) então
10:             v.dist ← u.dist + w(u,v)
11:             v.pai ← u
```

O algoritmo de Dijkstra começa definindo uma distância inicial para cada nó do grafo (linhas 1-3). A distância para todos os outros nós é definida como infinita, com exceção do nó de origem, que é definida como 0 (linha 4).

O laço principal do algoritmo (linhas 6-11) é executado enquanto ainda houver vértices não visitados em Q. Nesse laço, o algoritmo seleciona o vértice u em Q com a menor estimativa de distância ($u.dist$) entre os vértices de Q. Em seguida, para cada vértice v adjacente a u (isto é, para cada vértice v que é conectado a u por uma aresta), o algoritmo verifica se o caminho para v passando por u é mais curto do que o caminho atual para v. Se for, o algoritmo atualiza a estimativa de distância de *v.dist* para o caminho mais curto e define o antecessor *v.pai* de v como u (linhas 9-11). Esse processo é repetido até que todos os vértices sejam visitados, e a partir de cada vértice $v \in G$, é possível reconstruir o caminho mais curto entre s e v, através da atributo *pai*.

Seja $|V|$ o número de vértices do grafo e $|E|$ o número de arestas do grafo. Em geral, o custo computacional de Dijkstra é $O(|V|^2)$, quando é usada matriz de adjacência para representação do grafo. A complexidade do algoritmo pode ser reduzida para $O((|V| + |E|) \log |V|)$, quando são usadas listas de adjacências para a represetação do grafo.

Figura 2 – Execução do algoritmo de Dijkstra.



Fonte: (CORMEN et al., 2009), pg. 658.

A Figura 2 representa uma execução do algoritmo de Dijkstra. Primeiramente são inicializadas as distâncias de cada vértice (item a). Na primeira iteração são atualizados o vértices adjacentes ao vértice de origem s (item b). Na segunda iteração, são atualizadas as distâncias dos vétices adjacentes a y (item c). Observe que y foi o vértice escolhido pois ele possui a menor distância entre os vértices não visitados do grafo $(y.dist = 5)$. Ainda nessa iteração, o valor da distância de t é atualizada para $t.dist = 8$, pois o caminho passando de y até t é menor que o caminho passando de s até t. Esse processo é repetido até que o algoritmo percorra todos os vértices não visitados no grafo, e consequentemente atualizando seus valores de distância (itens d-f). Ao terminar a execução do algoritmo temos os valores de todos os menores caminhos de cada vértice a partir do vértice de origem s. Por exemplo, o menor caminho de s até z tem um custo de 7, pois $z.dist = 7$. Temos também que o menor caminho de s até z é $p = \langle s, y, z \rangle$, pois $z.pai = y$ e $y.pai = s$.

4 METODOLOGIA

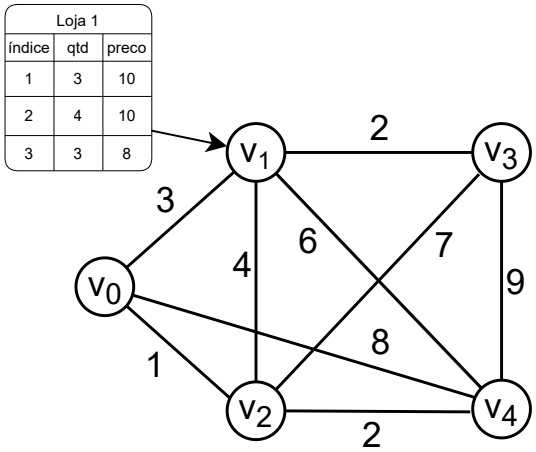
A presente solução consiste em encontrar a melhor rota para comprar uma lista de produtos em redes de lojas, levando em consideração a distância entre as lojas, a disponibilidade dos produtos e seus preços. Para representar o problema de busca de uma lista de produtos, é necessário modificar a estrutura do grafo G incluindo a lista de produtos de cada loja. Suponha que os produtos estejam organizados por indice de tal maneira a poder acessar o Produto 1 por $i = 1$. Considerando o grafo representado por $G = (V, E)$, para cada $v \in V$ representando uma loja, têm-se os seguintes atributos:

- $v.precototal$: representa o preço total dos produtos comprados no vértice v.
- $v.dist$: representa a distância do vértice de origem s até o vértice v.
- $v.pai$: representa o nó antecessor a v no menor caminho do vértice de origem s até v.

- $v.prod[]$: é uma lista que contém informações sobre os produtos presentes no vértice v. Para cada produto i na lista de produtos, são armazenadas as seguintes informações:
  - $v.prod[i]$: representa o produto de índice i no vértice v.
  - $v.prod[i].qtd$: representa a quantidade do produto i no vértice v.
  - $v.prod[i].preco$: representa o preço da unidade do produto i no vértice v.
  - $v.prod[i].qtd\_comprado$: representa a quantidade que foi comprada do produto i no vértice v. Caso o produto não tenha sido comprado, esse valor é igual a 0.

Também será criada uma nova entrada para a chamada do problema, que se trata do conjunto $usuario = \{pr_1, pr_2, pr_3, \ldots, pr_m\}$. Cada $pr_i \in usuario$ contém dois atributos: o número do índice do produto a ser comprado ($pr_i.id$) e a quantidade a ser comprada desse produto pelo usuário ($pr_i.qtd$).

Figura 3 – Grafo representando uma lista de lojas e a demanda de um usuário.



Fonte: Elaborado pelos autores.

O grafo da Figura 3 é representado por $G = (V, E)$, onde $V = (v_1, v_2, v_3, v_4, v_5)$. Cada loja do grafo tem uma lista de três produtos com suas quantidades e preços. Por exemplo, na vértice v_1, temos que a quantidade do produto 1 é $v_1.prod[1].qtd = 3$ e seu preço é $v_1.prod[1].preco = 10$. Já a demanda do usuário é definida por $usuario = (pr_1, pr_2, pr_3)$, onde $pr_1.id = 1$, $pr_1.qtd = 3$, $pr_2.id = 2$, $pr_2.qtd = 2$, $pr_3.id = 3$, $pr_3.qtd = 2$.

Dois problemas são tratados nesse trabalho, e consequentemente dois métodos são implementados para solucioná-los. O primeiro método (Método Simplificado) consiste em buscar em apenas uma única loja todos os produtos, sem levar em conta a quantidade de estoque dos produtos das lojas. Esse método é a versão mais simples do problema, que considera que todas as lojas têm estoque infinito de produtos e que só precisamos buscar por uma loja. Assim, o algoritmo retorna a loja com o menor custo total para comprar todos os produtos da lista. Observe que método não leva em conta encontrar um produto mais barato em uma loja e juntar com outro produto mais barato em outra loja, para fazer uma menor soma do custo total.

O segundo método (Método Generalizado) leva em conta a falta de estoque em algumas lojas e a necessidade de buscar produtos em várias lojas. Nesse método, buscamos os produtos mais baratos em função do menor caminho, levando em conta a distância entre as lojas e os preços unitários dos produtos. Para isso, o algoritmo usa o conceito de caminho de custo mínimo, que é o menor custo necessário para comprar a lista de produtos dentro do grafo. O algoritmo retorna uma série de vértices (lojas) que contém a melhor forma de comprar a lista de produtos definida pelo usuário, seguindo o caminho de menor custo. Dessa forma, é possível encontrar a melhor rota para comprar todos os produtos com o menor custo possível.

4.1 Método Simplificado

Como dito anteriormente, nesse método (Algoritmo 2) é considerada apenas uma loja como objetivo do problema. Para isso, é assumido que todas as lojas tenham quantidade suficiente de todos os produtos em estoque (i.e., $v.prod[i].qtd = \infty$ para todo i). A ideia do método é modificar o algoritmo de Dijkstra para calcular os custos benefícios dos vértices durante cada iteração. O custo benefício consiste, da distância entre o vértice e seu antecessor e do somatório dos preços dos produtos no vértice atual. Ao final da execução do algoritmo, será calculado o vértice de melhor custo benefício para comprar os produtos, assim resolvendo o problema de comprar todos os produtos em uma única loja.

Algoritmo 2 MÉTODO-SIMPLIFICADO($G, usuario, s$)

```
1:  para cada v ∈ G.V faça                          ▷ Inicialização das distâncias dos vértices
2:      v.dist ← ∞
3:      v.pai ← NIL
4:      para cada pr ∈ usuario faça                  ▷ Inicialização dos preços totais
5:          v.precototal ← v.precototal + v.prod[pr.id].preco
6:  s.dist ← 0
7:  s.precototal ← ∞
8:  melhor ← s                                       ▷ Vértice correspondente à solução do algoritmo
9:  Q ← G.V                                          ▷ Fila de prioridade com todos os nós
10: enquanto Q ≠ ∅ faça
11:     u ← EXTRACT-MIN(Q)                           ▷ Remoção do nó com menor distância de Q
12:     para cada v ∈ G.ADJ(u) faça                  ▷ Atualização das distâncias
13:         se v.dist > u.dist + w(u,v) então
14:             v.dist ← u.dist + w(u,v)
15:             v.pai ← u
16:         se melhor.precototal + melhor.dist > v.dist + v.precototal então
17:             melhor ← v                           ▷ Atualização da nova melhor loja
18: retorne melhor                                   ▷ Retorno da melhor loja para comprar os produtos
```

O algoritmo MÉTODO-SIMPLIFICADO($G, usuario, s$) começa definindo uma distância inicial para cada nó do grafo (linhas 1-3) e o preço total ($v.precototal$) dos produtos na loja como a soma dos produtos do vértice (linhas 4-5). A distância para todos os outros nós é definida como

infinita, com exceção do nó de origem, que é definida como 0 (linha 6) e o preço total do nó de origem como infinito. Também define o melhor vértice para comprar os produtos como sendo s.

O laço principal do algoritmo (linhas 10-17) é executado enquanto ainda houver vértices não visitados em Q. Nesse laço, o algoritmo seleciona o vértice u em Q com a menor estimativa de distância $(u.dist)$ entre os vértices Q. Em seguida, para cada vértice v adjacente a u, o algoritmo verifica se o caminho para v passando por u é mais curto do que o caminho atual para v. Se for, o algoritmo atualiza a estimativa de distância de $v.dist$ para o caminho mais curto e define o antecessor $v.pai$ de v como u (linhas 12-15). Neste mesmo laço, é verificado se o vértice adjacente a u tem um custo menor que o melhor vértice visitado. Se for define *melhor* como v (linhas 16-17). Esse processo é repetido até que todos os vértices sejam visitados, e a partir de cada vértice em v em G, é possível reconstruir o caminho mais curto entre s e v, através do atributo $v.pai$. O vértice com melhor custo benefício está definido na variável *melhor* ao final da execução do algoritmo.

O custo desse algoritmo pode ser analisado pela soma dos dois laços do algoritmo. O laço das linhas 1-5 tem complexidade de $O(|V| \times |usuario|)$ e o laço das linhas 10-17 tem complexidade $O(|V| \times |E|)$. Portanto, a complexidade de tempo total do algoritmo MÉTODO-SIMPLIFICADO$(G, usuario, s)$ é de $O(|V| \times |usuario| + |V| \times |E|)$. Caso seja considerado $usuario = |V|$, pode-se definir a complexidade do algoritmo como $O(|V|^2)$.

Para uma melhor compreensão do código, considere os seguintes dados abaixo como os dados de entrada simulados para o problema. A Figura 4 apresenta a lista de produtos que o usuário deseja comprar e ao lado a lista de lojas com os seus respectivos produtos, incluindo quantidade e preço. Observe que a loja 0 representa o nó inicial, ou a residência do usuário. Logo, as quantidades de produtos são definidas como 0 e seus preços como infinito. A quantidade de produtos nas outras lojas são definidas como infinito, pois para esse problema é assumido que todas as lojas possuem os produtos em estoque.

Figura 4 – Lista de produtos a serem comprados e produtos disponíveis nas lojas.

usuario	
id	qtd
1	3
2	3
3	2

Loja 0		
índice	qtd	preco
1	0	∞
2	0	∞
3	0	∞

Loja 1		
índice	qtd	preco
1	∞	10
2	∞	10
3	∞	8

Loja 2		
índice	prod	preco
1	∞	10
2	∞	10
3	∞	10

Loja 3		
índice	qtd	preco
1	∞	10
2	∞	1
3	∞	1

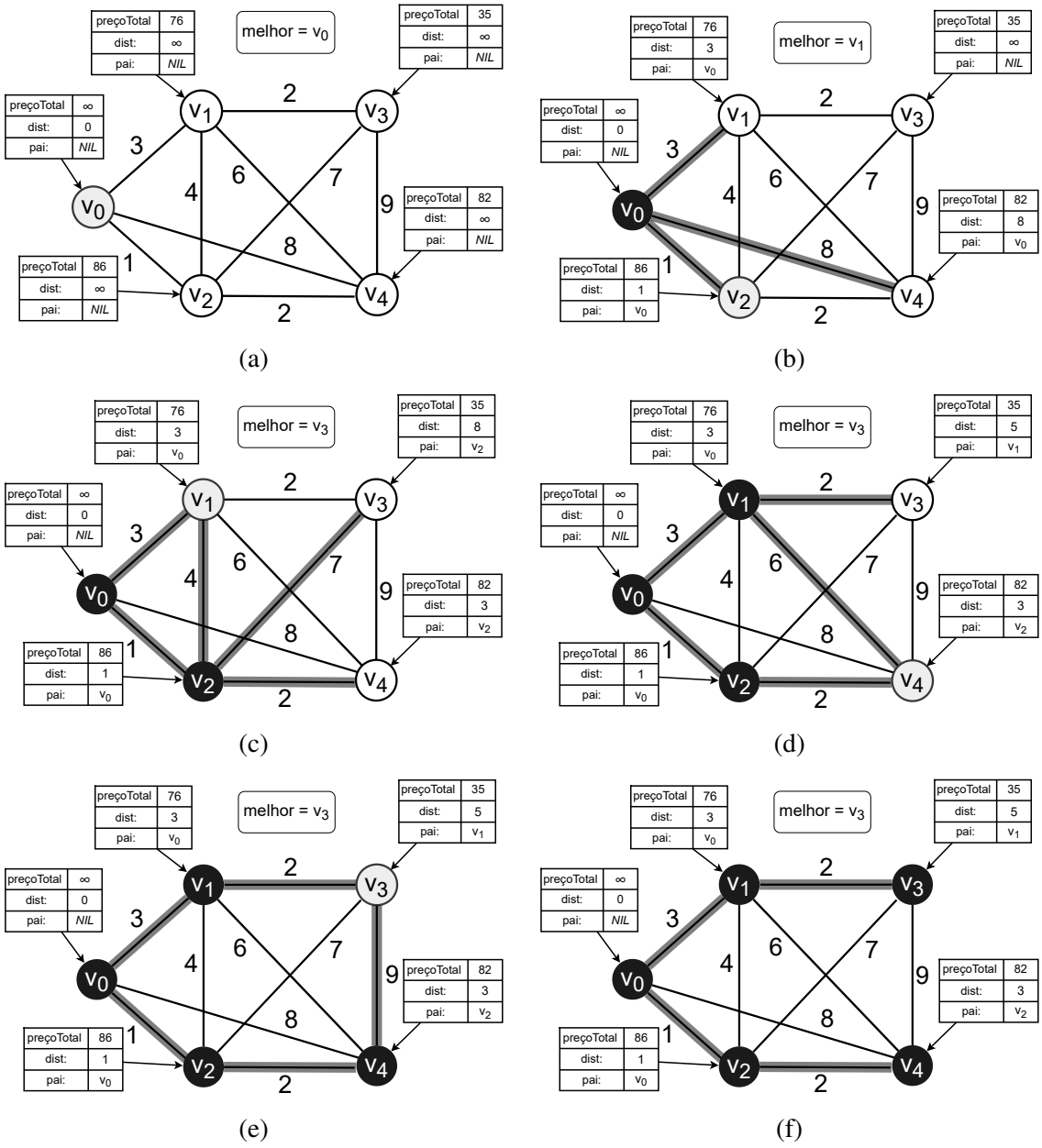
Loja 4		
índice	qtd	preco
1	∞	12
2	∞	10
3	∞	8

Fonte: Elaborado pelos autores.

A Figura 5 representa a execução do algoritmo de Metodo Simplificado. Primeiramente são inicializadas as distâncias de cada vértice (item a) e definir *melhor* como o vértice inicial s. Na primeira iteração são atualizados os vértices adjacentes ao vértice de origem s (v_0) e atualiza *melhor* para o vértice v_1 (item b). Perceba que o melhor vértice (*melhor*) foi atualizado de v_0 para v_1, já que $79 = v_2.dist + v_2.precotal < v_0.dist + v_0.precotal = \infty$. Na segunda iteração, são atualizadas as distâncias dos vértices adjacentes a v_2 (item c). Observe que v_2 foi o vértice escolhido pois ele possui a menor distância entre os vértices não visitados do grafo $(v_0.dist = 1)$.

Ainda nessa iteração, o valor da distância de v_3 é atualizada para $v_3.dist = 8$, sendo o caminho passando de v_2 até v_3. Igualmente é verificado se o custo total do vértice v_3 é menor que o melhor anterior, no caso o vértice v_1. É verificado então que $43 = v_3.dist + v_3.precototal < melhor.dist + melhor.precototal = v_1.dist + v_1.precototal = 79$.

Figura 5 – Execução do Algoritmo



Fonte: Elaborado pelos autores.

Esse processo é repetido até que o algoritmo percorra todos os vértices não visitados no grafo, e consequentemente atualizando seus valores de distância (itens d-f). Ao terminar a execução do algoritmo temos os valores de todos os menores caminhos de cada vértice a partir do vértice de origem s e o vértice que representa a solução do problema, no caso o vértice v_3.

De acordo com o problema, o menor caminho de s até v_3 tem um custo de 5, pois $v_3.dist = 5$. Temos também que o menor caminho de s até v_3 é $p = \langle s, v_1, v_3 \rangle$, pois $v_3.pai = v_1$ e $v_1.pai = s$.

4.2 Método Generalizado

Este método é a generalização do problema, onde existe uma demanda de produtos pelo usuário e as lojas podem ou não conter a quantidade de produtos necessários para completar o pedido. Assim como no método simplificado, podemos definir a melhor rota para compra como o caminho de s a v_i com o melhor custo benefício. Entretanto, neste método teremos de considerar as combinações de caminhos e produtos disponíveis nas lojas.

Considere a soma dos produtos comprados por seu preço e suas respectivas quantidades no caminho. Definimos o custo como o somatório de cada produto comprado pelo seu preço $(v_i.prod[j].preco * v_i.prod[j].qtd\_comprado)$, onde j se trata de cada índice de produto em *usuario*. O custo total da compra de cada loja i trata da soma dos produtos comprados nesta loja, e este valor será guardado em $v_i.precototal$. O melhor caminho será aquele onde essa relação de $preo + distncia$ seja a menor possível. Considere o caminho $p = \langle v_1, v_2, ..., v_n \rangle$,

$$CustoTotal(p, usuario) = w(p) + \sum_{v=v_1}^{v_n} \left(\sum_{j=1}^{|usuario|} v.prod[j].preco * v.prod[j].qtd\_comprado \right).$$

Definido o custo total, o caminho mais curto em nosso problema pode ser dado por:

$$\delta(u,v) = \begin{cases} \min\{CustoTotal(p, usuario) : u \overset{p}{\rightsquigarrow} v\} & \text{se existe um caminho de } u \text{ para } v, \\ \infty & \text{caso contrário.} \end{cases}$$

A ideia acima é percorrer os vértices intermediários para todos os vértices do grafo a fim de definir o melhor caminho. De tal maneira podemos definir, para o método generalizado, um novo atributo para cada vértice, chamado *v.lista_usuario*, contendo a lista de produtos do usuário do melhor caminho até o vértice v. Iremos garantir essa condição da seguinte maneira: Para cada vértice intermediário entre s e v, verificar se o caminho passando por $s \overset{v_i}{\rightsquigarrow} v$ é melhor que o último caminho conhecido de v, se sim, v e substituído por esse caminho.

Sendo $v \in V$, considere a função COPIAR(v) como um método da linguagem para criar cópias dos vértices. Ela se trata de uma função genérica e irá depender da linguagem em que for implementada. Assuma que a função COPIAR(v) retorna uma cópia contendo todos os atributos de v. O custo dessa função será igual ao número de atributos de v. Como em v existe uma lista variável chamada $v.prod[]$, seu custo será dado pelo número de produtos desta lista (tamanho de *usuario*), logo seu custo será $O(|usuario|)$.

Na sequência, é definida a função ATUALIZA-LISTA(v_i, v_j) (Algoritmo 3), que recebe o vértice intermediário v_i e o vértice a ser melhorado v_j. O algoritmo cria uma cópia de v_j na variável v' e atribui a nova distância passando por v_i, então copia a $v_i.lista\_usuario$ para $v'.lista\_usuario$ (linhas 1-4). Ele cria uma cópia do caminho de v_i e atribui a $v'.pai$ (linhas 6-10).

Para cada produto desejado por $v'.lista\_usuario$ no caminho, verificar o produto correspondente no $v'.prod$, se a demanda não for saciada pela quantidade, adicionar o possível, se sobrar produtos na loja, é verificado se é possível substituir algum produto já armazenado. Ao final do algoritmo é garantido que o caminho contém a melhor maneira de comprar os produtos p da lista do usuário. O algoritmo então define o custo total do caminho e atribui em $v'.precototal$ e retorna v' como resposta. Seu custo pode ser definido pelos laço da linhas 6-9, com custo $O(|V|^2)$ e linhas 10-25 com custo $O(|V|^2)$, ou seja, o custo total é $O(|V|^2)$.

Algoritmo 3 ATUALIZA-LISTA(v_i, v_j)

```
1:  v' ← COPIAR(v_j)                                   ▷ Criação de uma copia do vertice v_j
2:  para cada p ∈ v_i.lista_usuario faça                ▷ Cópia da lista de v_i em v'
3:      v'.lista_usuario[p.id].qtd ← p.qtd
4:  v'.dist ← v'.dist + W(v_i, v_j)                     ▷ Definição da nova distância
5:  v'' ← v'                                            ▷ Cópia de v' para ser usada como caminho intermediário
6:  enquanto v_i ≠ NIL faça
7:      v''.pai ← COPIAR(v_j)
8:      v'' ← v''.pai
9:      v_i ← v_i.pai
10: para cada pr ∈ v'.lista_usuario faça                ▷ Atualização de cada produto em v'
11:     se v'.prod[pr.id].qtd < pr.qtd então            ▷ Teste se a qtd de produtos é menor que na lista
12:         v'.prod[pr.id].qtd_comprado ← v'.prod[pr.id].qtd   ▷ Inserção como comprado
13:         pr.qtd ← pr.qtd − v'.prod[pr.id].qtd
14:     senão                                           ▷ Substituição de produtos no caminho intermediário se sobrar de v'
15:         v'' ← v'.pai
16:         resto ← v'.prod[pr.id].qtd − pr.qtd
17:         enquanto v'' ≠ NIL faça                     ▷ Travessia do caminho intermediário
18:             se v''.prod[pr.id].qtd_comprado > 0 então   ▷ Teste caso tenha comprado
19:                 se v''.prod[pr.id].preco + v''.dist > v'.prod[pr.id].preco + v'.dist então
20:                     n ← v''.prod[pr.id].qtd_comprado
21:                     v''.prod[pr.id].qtd_comprado ← v''.prod[pr.id].qtd_comprado − resto
22:                     resto ← resto − n
23:             v'' ← v''.pai
24:         v'.prod[pr.id].qtd_comprado ← v'.prod[pr.id].qtd − resto
25:         pr.qtd ← 0
26: v'.precototal ← CUSTO-CAMINHO(v')                   ▷ Cálculo do custo total do caminho
27: retorne v'
```

Na sequência é definida a função para calcular o custo do caminho, chamada CUSTO-CAMINHO(v) (Algoritmo 4). Essa função percorre o caminho de v e calcula o somatório da quantidade de produtos comprados vezes o seu preço na loja em que foi comprado. Se o caminho não tiver completado a lista de compras, definimos o caminho como ∞. O custo dessa função é o mesmo do laço das linhas 8-12 que é de $O(|V|^2)$.

Algoritmo 4 CUSTO-CAMINHO(v)

```
1: SUM ← SUM + v.dist                          ▷ Incremento do custo do caminho à SUM
2: para cada pr ∈ v.usuario faça               ▷ Teste se foi possível completar a lista do usuario
3:     se pr.qtd > 0 então
4:         retorne ∞                           ▷ Retorna ∞ caso não
5: enquanto v ≠ NIL faça
6:     para cada p ∈ v.prod faça
7:         se p.qtd_comprado > 0 então
8:             SUM ← SUM + (p.qtd_comprado * p.preco)   ▷ Somatório do custo na loja
9:     v ← v.pai
10: retorne SUM
```

Por fim, é definido o METODO-GENERALIZADO($G, s, usuario$) (Algoritmo 5). Para cada caminho v_i é verificar se o caminho de v_j passando por v_i pode ser melhorado. Se o caminho v_j for melhorado pelo vértice intermediário v_i, ele irá substituir o caminho v_j por v'_j.

Algoritmo 5 METODO-GENERALIZADO($G, s, usuario$)

```
1: para cada v ∈ G.V faça                      ▷ Inicialização das distâncias dos vértices
2:     v.dist ← ∞
3:     v.pai ← NIL
4:     v.precototal ← ∞
5:     v.lista_usuario ← ∅                     ▷ Inicialização da lista de compras do caminho como vazia
6:     para cada p ∈ usuario faça
7:         p' ← p                              ▷ p' recebe uma copia de p
8:         v.lista_usuario ← v.lista_usuario ∪ {p'}   ▷ Atribuição da cópia da lista
9: s.dist ← 0
10: v_r ← s                                    ▷ Definição do vértice de menor custo como s
11: para cada v_i ∈ G.V faça                   ▷ Travessia dos vértices intermediários
12:     para cada v_j ∈ G.ADJ(v_i) faça
13:         esta ← false
14:         caminho ← v_i
15:         enquanto v ≠ NIL faça
16:             se v = v_j então
17:                 esta ← true
18:             v ← v.pai
19:         se esta ≠ true então
20:             v'_j ← ATUALIZA-LISTA(v_i, v_j)   ▷ Criação do caminho até v_j passando por v_i
21:             se v'_j.precototal ≤ v_j.precototal então
22:                 v_j.pai ← v'_j.pai
23:                 v_j.dist ← v'_j.dist
24:                 v_j.precototal ← v'_j.precototal
25:                 v_j.lista_usuario ← v'_j.lista_usuario
26:             se v_j.precototal < v_r.precototal então   ▷ Teste se o caminho v_j é melhor
27:                 v_r ← v_j
28: retorne v_r
```

O algoritmo METODO-GENERALIZADO($G, s, usuario$) começa definindo a distância inicial para cada nó do grafo (linhas 1-5). A distância para todos os outros nós e peso total são definidos como infinito, com exceção do nó de origem, cujo a distância é definida como 0 (linha 9). Nas linhas 6-8 é criada a uma cópia da lista *usuario* para cada vértice.

O laço externo do algoritmo é executado para todo vértice v_i em $G.V$ e o interno também para para todos vértice v_j adjacentes a v_i (linhas 11-27). Se o vértice v_j não apareceu no caminho de v_i, então a função ATUALIZA-LISTA(v_i, v_j) é chamada para criar o caminho de v_j passando por v_i e é guardada em v'_j (linha 20). Então é realizada a verificação $v'_j.precototal < v_j.precototal$, para verificar se existe um caminho intermediário melhor que o caminho até v_j (linhas 21-27). Além disso, é verificado se o vértice v_j tem um custo total menor que o melhor vértice já armazenado em v_r, se sim definir v_r como o vértice v_j. Esse processo é repetido até que se tenha andado em todos os vértices intermediários. Ao final desse laço (linhas 11-27) a variável v_r guarda o vértice de melhor custo, assim o caminho de melhor custo para comprar a lista de produtos no grafo.

O custo desse algoritmo pode ser analisado a soma dos laços e das funções do algoritmo METODO-GENERALIZADO($G, s, usuario$), onde o laço das linhas 1-8 tem custo $O(|V^2|)$, o laço das linhas 11-27 tem custo $O(|E| * |V|^3)$. Portanto, a complexidade de tempo total do algoritmo METODO-GENERALIZADO($G, s, usuario$) é $O(|E| * |V|^3)$.

A Figura 6 apresenta um exemplo de uma lista de produtos que o usuário deseja comprar e ao lado a lista de lojas com os seus respectivos produtos, incluindo quantidade e preço. Observe que a loja 0 representa o nó inicial, ou a residência do usuário. Logo, as quantidades de produtos são definidas como 0 e seus preços como infinito. A quantidade de produtos nas outras lojas são definidas de acordo com as quantidades disponíveis, pois para esse problema é assumido que todas as lojas podem não possuir todos os produtos em estoque, ou ser insufuciente para a demanda do usuario.

Figura 6 – Dados das lojas.

usuario	
id	qtd
1	3
2	3
3	2

Loja 0			
índice	qtd	preco	qtd_C
1	0	∞	0
2	0	∞	0
3	0	∞	0

Loja 1			
índice	qtd	preco	qtd_C
1	3	10	0
2	4	10	0
3	3	8	0

Loja 2			
índice	qtd	preco	qtd_C
1	1	10	0
2	0	∞	0
3	1	10	0

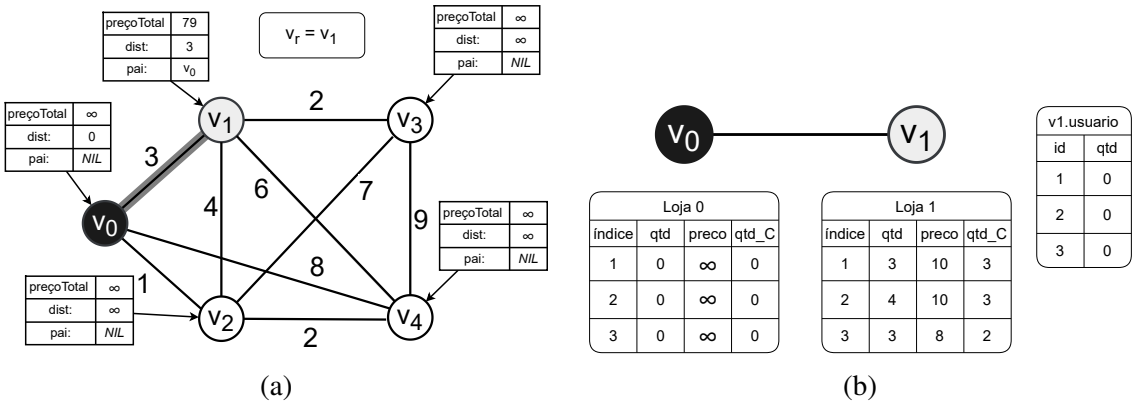
Loja 3			
índice	qtd	preco	qtd_C
1	8	10	0
2	7	1	0
3	7	1	0

Loja 4			
índice	qtd	preco	qtd_C
1	7	12	0
2	5	10	0
3	5	8	0

Fonte: Elaborado pelos autores.

Primeiramente são inicializadas as distâncias de cada vértice, em que todos têm distância e *precototal* infinita, exceto o s (v_0) que tem distância 0. A Figura 7 apresenta os dados durante a primeira interação, que verifica os caminhos passando por v_0. O item (a) mostra a verificação de v_1, onde verifica que o caminho de v_1 melhorou ao passar pelo v_0, ou seja a caminho e conseguiu zerar a demanda e a soma dos produtos no caminho foi 79 (item b). O vértice v_r foi definido como v_1, já que $79 = v_1.precototal < v_r.precototal = \infty$.

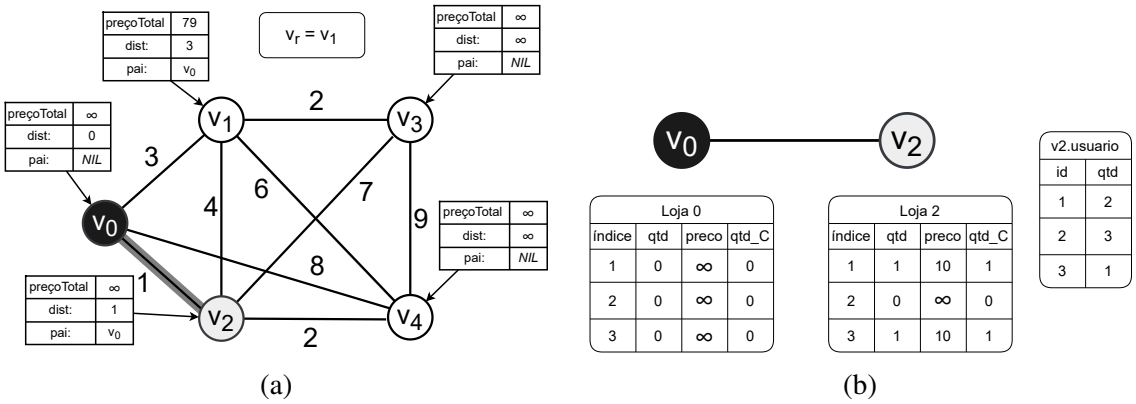
Figura 7 – Caminho de v_0 para v_1.



Fonte: Elaborado pelos autores.

Observe que o caminho de vértice v_2 passando por v_0 (Figura 8) não conseguiu atender a demanda (item b), então $v_2.pesototal$ foi definido como ∞ e seu pai como v_0 (item a). Logo, v_r não foi alterado, já que $\infty = v_2.pesototal > v_r.pesototal = 79$.

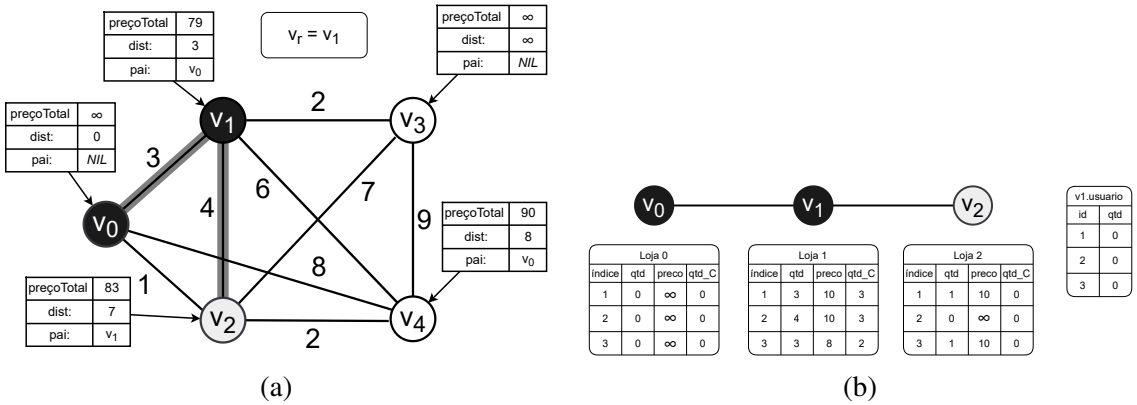
Figura 8 – Caminho de v_0 para v_1.



Fonte: Elaborado pelos autores.

A Figura 9 mostra o estado da segunda iteração de v_i. Considerando o vértice v_2 tendo v_1 como vértice intermediario, v_2 conseguiu completar a demanda do usuario (zerar a lista de produtos) (item b) e o caminho foi melhorado de $\infty > 83$. Então v_2 recebe o novo caminho tendo v_1 como intermediario, seu *precototal* foi definido como 83 e seu *pai* como v_1 (item a).

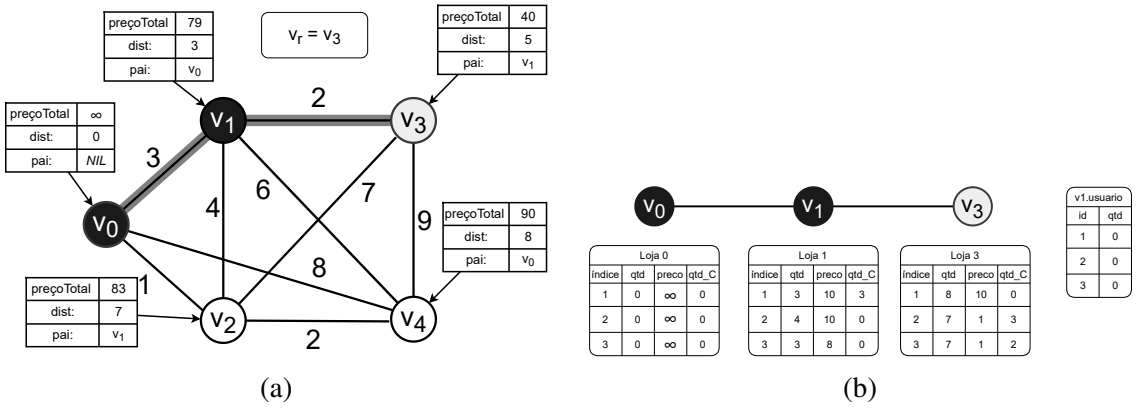
Figura 9 – Caminho de v_1 para v_2.



Fonte: Elaborado pelos autores.

Durante a verificação do vértice v_3 passando por v_1 (Figura 10) ele verifica que v_3 é acessível tendo v_1 como intermediário e atualiza o caminho, onde $v_3.pai$ foi definido como v_1, e $v_3.precototal$ foi definido como 40 (item c). Observe que o caminho que passa por v_1 foram removidos os produtos 1 e 2 de v_1 e adicionados de v_3, assim melhorando o caminho (item d). Então v_r foi definido como v_3, já que $40 = v_3.precototal < v_1.precototal = 79$.

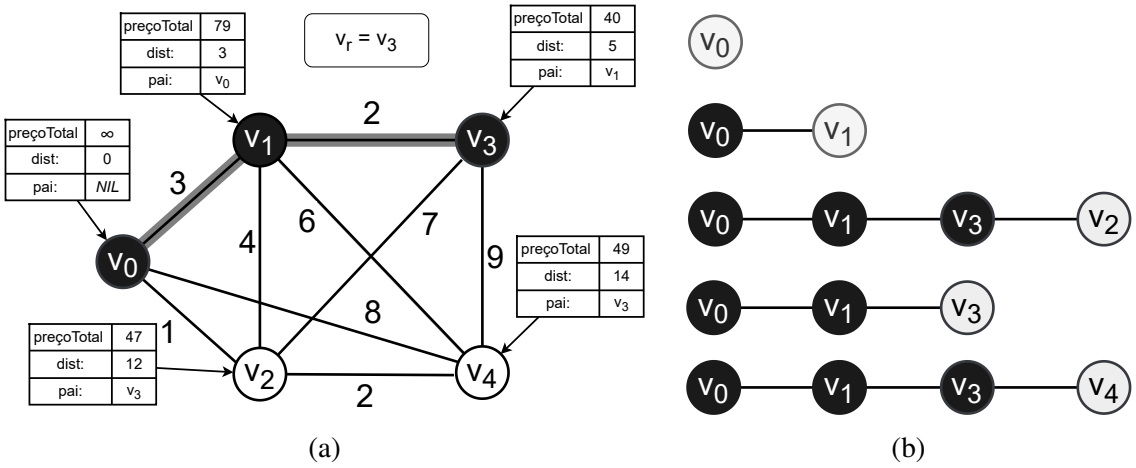
Figura 10 – Caminho de v_1 para v_3.



Fonte: Elaborado pelos autores.

Esse processo é repetido até que o algoritmo percorra todos os vértices intermediários possíveis para cada vértice no grafo, e consequentemente atualizando seus valores de distância, seus caminhos e custos totais. Ao terminar a execução do algoritmo temos os valores de todos os menores caminhos de cada vértice a partir do vértice de origem s (Figura 11) e o vértice que representa a solução do problema, no caso o vértice v_3. De acordo com o problema, o menor caminho de s até v_3 tem um custo de 40, cujo a soma dos produtos comprados somados à distancia percorrida é a menor entre todos os caminhos.

Figura 11 – Grafo com os caminhos obtidos.



Fonte: Elaborado pelos autores.

5 RESULTADOS OBTIDOS

Para os resultados foram realizadas as implementações dos algoritmos na plataforma Google Colab[1] na linguagem Python, podendo ser acessada pelos links nos anexos. Para a implementação dos métodos simplificado e generalizado foram usados cenarios hipoteticos, a entrada mostrada nas Figuras 12 e 13 representam o cenário onde o usuario deseja comprar os produtos 1,2 3 em uma cidade com 4 lojas. Observe também que os dados são os mesmos utilizados nas seções 4.1 e 4.2, logo espera-se que o resultado seja o mesmo que o encontrado nestas seções, respectivamente $v_3.pesototal + v_3.dist = 40$ e $v_3.pesototal = 40$.

Figura 12 – Dados de entrada.

```
v0 = V(id = 0)
v0.PROD.append(Produto(1,0, 9999999999999))
v0.PROD.append(Produto(2,0, 9999999999999))
v0.PROD.append(Produto(3,0, 9999999999999))
v0.PROD.append(Produto(4,0, 9999999999999))

v1 = V(id = 1)
v1.PROD.append(Produto(1,3, 10.0))
v1.PROD.append(Produto(2,4, 10.0))
v1.PROD.append(Produto(3,3, 8.0))
v1.PROD.append(Produto(4,2, 8.0))

v2 = V(id = 2)
v2.PROD.append(Produto(1,1, 10.0))
v2.PROD.append(Produto(2,0, 9999999999999))
v2.PROD.append(Produto(3,1, 10.0))
v2.PROD.append(Produto(4,1, 10.0))

v3 = V(id = 3)
v3.PROD.append(Produto(1,8, 10.0))
v3.PROD.append(Produto(2,7, 1.0))
v3.PROD.append(Produto(3,7, 1.0))
v3.PROD.append(Produto(4,9, 1.0))
```

Fonte: Elaborado pelos autores.

[1] https://colab.research.google.com/

Observe que o número 9999999999999 foi utilizado para representar o valor infinito.

Figura 13 – Dados do grafo

```
v4 = V(id = 4)
v4.PROD.append(Produto(1,7, 12.0))
v4.PROD.append(Produto(2,5, 10.0))
v4.PROD.append(Produto(3,5, 8.0))
v4.PROD.append(Produto(4,16, 8.0))

# Criando Es
a0 = E(v0, v1, 3)
a1 = E(v0, v2, 1)
a2 = E(v0, v4, 8)
a3 = E(v1, v2, 4)
a4 = E(v1, v3, 2)
a5 = E(v1, v4, 6)
a6 = E(v2, v3, 7)
a7 = E(v2, v4, 2)
a8 = E(v3, v4, 9)
# Criando grafo
grafo1 = Grafo([v0, v1, v2, v3 ,v4], [a0, a1, a2, a3, a4, a5, a6, a7, a8])

# Criando a lista de demanda do usuario
up1 = Produto(1,3, 0)
up2 =Produto(2,3, 0)
up3 = Produto(4,2, 0)
usuario1 = USERD([up1,up2,up3])
```

Fonte: Elaborado pelos autores.

Para computar a resposta é feita a chamada do Algoritmo METODO-SIMPLIFICADO(grafo, 0, user) que se encontra no link disponibilizado nos anexos do arquivo. A saída do algoritmo é a modificação do grafo geral e o retorno do vértice com menor custo benefício. A Figura 14 apresenta a saída da execução do Método Simplificado para cada vértice, onde assim como definido na metodologia 4.1, a resposta foi o vértice v_3, assim o resultado esperado foi igual ao resultado obtido, tal que $v_3.pesototal + v_3.dist = 40$ é igual ao obtido durante a explicação do método na seção 4.1. Isso mostra que o comportamento do algoritmo implementado correspondeu a lógica definida nas seções anteriores.

Figura 14 – Resultado da execução do Método Simplificado.

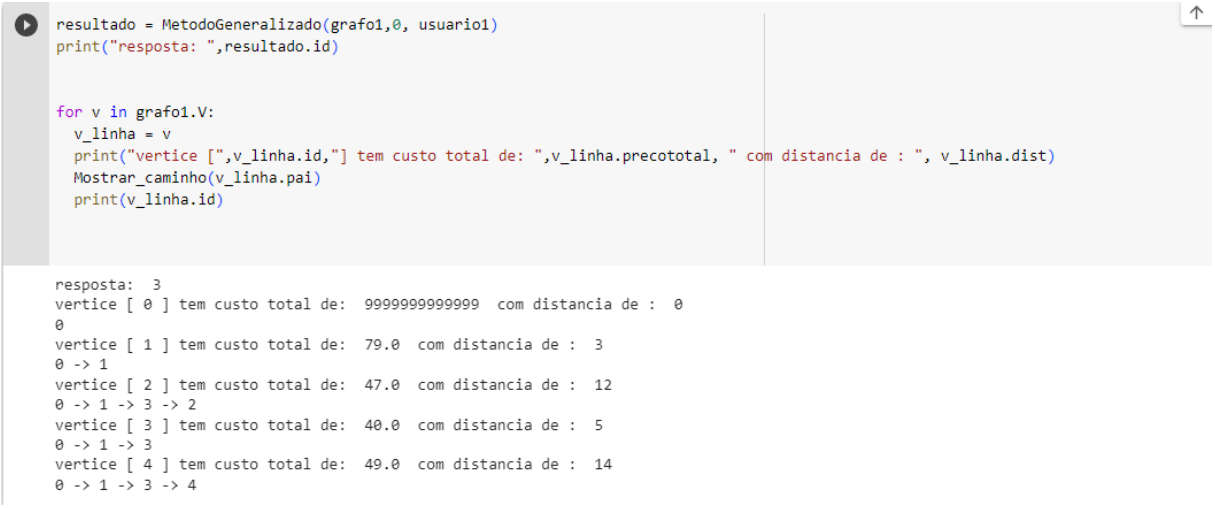
```
resultado = MetodoSimplificado(grafo,0,usuario)
print("resposta: ",resultado.id)


for v in grafo.V:
  v_linha = v
  print("vertice [",v_linha.id,"] tem custo total de: ",v_linha.precototal, " com distancia de : ", v_linha.dist)
  Mostrar_caminho(v_linha.pai)
  print(v_linha.id)
```

```
resposta:  3
vertice [ 0 ] tem custo total de:  79999999999992  com distancia de :  0
0
vertice [ 1 ] tem custo total de:  76.0  com distancia de :  3
0 -> 1
vertice [ 2 ] tem custo total de:  86.0  com distancia de :  1
0 -> 2
vertice [ 3 ] tem custo total de:  35.0  com distancia de :  5
0 -> 1 -> 3
vertice [ 4 ] tem custo total de:  82.0  com distancia de :  3
0 -> 2 -> 4
```

Fonte: Elaborado pelos autores.

A Figura 15 apresenta a saída da computação do grafo para o Método Generalizado, onde é chamada a função METODO-GENERALIZADO(grafo, 0, usuario), que pode ser vista no link disponibilizado em anexo. Perceba que foi utilizada a mesma entrada que a utilizada no método simplificado, já que os valores utilizados pelo método simplificado simplesmente assumiram quantidade igual ∞. O algoritmo retorna o vértice de melhor custo benefício e a modificação dos caminhos no grafo onde cada vértice tem seu melhor caminho para comprar a lista do usuário. Os dados retornados são para cada caminho é igual aos dados esperados na metodologia 4.2, tal que o vértice com melhor caminho foi v_3, e o vértice esperado era v_3, onde $v_3.pesototal = 40$, o mesmo que o encontrado no método 4.2.

Figura 15 – Resultado da execução do Metodo Generalizado.



```
resultado = MetodoGeneralizado(grafo1,0, usuario1)
print("resposta: ",resultado.id)


for v in grafo1.V:
  v_linha = v
  print("vertice [",v_linha.id,"] tem custo total de: ",v_linha.precototal, " com distancia de : ", v_linha.dist)
  Mostrar_caminho(v_linha.pai)
  print(v_linha.id)
```

```
resposta:  3
vertice [ 0 ] tem custo total de:  9999999999999  com distancia de :  0
0
vertice [ 1 ] tem custo total de:  79.0  com distancia de :  3
0 -> 1
vertice [ 2 ] tem custo total de:  47.0  com distancia de :  12
0 -> 1 -> 3 -> 2
vertice [ 3 ] tem custo total de:  40.0  com distancia de :  5
0 -> 1 -> 3
vertice [ 4 ] tem custo total de:  49.0  com distancia de :  14
0 -> 1 -> 3 -> 4
```

Fonte: Elaborado pelos autores.

Observe que os custos para os caminhos no grafo mudaram de um algoritmo para o outro devido a entrada ser diferente. Para o METODO-GENERALIZADO(grafo, 0, usuario) as lojas têm uma quantidade de produtos variáveis assim podendo encarecer ou baratear o caminho final, o que diferencia de METODO-SIMPLIFICADO(grafo, 0, usuario) onde a quantidade de produtos na loja é ∞.

Com base no que foi apresentado durante o trabalho, podemos perceber que o método 4.1 tem uma complexidade menor que o método 4.2, $O|V^2| < O|V^4|$, porém se restringe a cenários restritos, onde se busca apenas uma loja com os produtos, já o segundo método, trabalha com múltiplas lojas e quantidades variáveis, assim trazendo otimização máxima. Em suma, o método simplificado 4.1 é melhor em termos de complexidade e tempo de execução, porém não satisfaz o problema de forma completa, já o método 4.2 apesar de mais complexo, satisfaz a problemática de forma mais completa, considerando problemas reais, como falta de produtos, e quantidades insuficientes.

6 CONCLUSÃO

Os resultados mostram que para este problema da compra de uma lista de produtos, os algoritmos podem ser resolvidos em tempo polinomial. Para isso, foram possíveis as criações de dois métodos. De maneira geral, o primeiro método mostrou ter uma complexidade de tempo mais eficiente que o segundo, porém com condições menos restritivas. O segundo método, apesar de ter maior custo computacional, aborda o problema de forma geral incluindo variáveis que representam melhor situações reais, como falta de produtos e melhoria por comprar em lojas diferentes, assim encontrando a melhor rota para comprar a lista de produtos.

Em conclusão, o estudo apresentado mostrou que o algoritmo de Dijkstra pode ser adaptado para solucionar problemas práticos do mundo real, como a determinação da melhor maneira de comprar uma lista de produtos em redes de lojas. Ambas as abordagens apresentaram resultados satisfatórios em termos de tempo de execução e complexidade de tempo, indicando que são úteis para problemas de tamanho moderado a grande, além de poder ser adaptado para qualquer problema que envolva problemas de grafos com listas de objetos nas arestas.

Para trabalhos futuros, há várias direções que podem ser exploradas com base neste artigo. Uma das possibilidades é a criação de uma aplicação que disponibiliza a funcionalidade desenvolvida como um site ou um aplicativo *mobile*. Isso permitiria que os usuários aproveitassem os algoritmos desenvolvidos aqui para melhorar suas próprias experiências. Além disso, seria interessante dedicar esforços para reduzir a complexidade dos algoritmos METODO-GERAL e METODO-SIMPLIFICADO. A simplificação desses métodos pode torná-los mais acessíveis e fáceis de serem implementados em diferentes contextos.

Outra linha de pesquisa promissora seria explorar algoritmos *all paths* para o problema da compra de uma lista de produtos, exemplo o algoritmo de Floyd Warshall. O algoritmo de Floyd-Warshall é amplamente conhecido por encontrar o caminho mais curto entre todos os pares de vértices em um grafo ponderado. Adaptar esse algoritmo para o contexto de compras permitiria encontrar o caminho mais eficiente para visitar as lojas e adquirir todos os produtos da lista, levando em consideração a distância entre as lojas, o tempo gasto em cada uma delas e outras restrições relevantes.

Em suma, os trabalhos futuros podem envolver o desenvolvimento de uma aplicação prática, a simplificação dos algoritmos propostos e a exploração de outros algoritmos de aplicação. Essas investigações podem ampliar o impacto dos algoritmos desenvolvidos neste artigo, tornando-os acessíveis a um público mais amplo e abrindo caminho para avanços significativos na área de Algoritmos em Grafos.

REFERÊNCIAS

ZHU, Z. et al. (Ed.). *Application of improved Dijkstra algorithm in intelligent ship path planning*. 33rd Chinese Control and Decision Conference (CCDC), Kunming, China: IEEE, 2021. 4926–4931 p.

CHEN, R. Dijkstra's shortest path algorithm and its application on bus routing. Atlantis Press, p. 321–325, may. 2022. ISSN 2352-5428.

CORMEN, T. H. et al. **Introduction to Algorithms**. 3rd. ed. Cambridge, Massachusetts: The MIT Press, 2009. ISBN 9780262533058.

CORREIA, S. P. **Otimização de rotas para a entrega de correspondências**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2019.

FLOYD, R. W. Algorithm 97: Shortest path. **Communications of the ACM**, v. 5, p. 345, 1962.

HART, P.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. Institute of Electrical and Electronics Engineers (IEEE), v. 4, n. 2, p. 100–107, 1968.

BHATTACHARYYA, S.; BANERJEE, J. S.; KÖPPEN, M. (Ed.). **Optimal Path Planning with Smart Energy Management Techniques Using Dijkstra's Algorithm**, v. 316. Human-Centric Smart Computing: Proceedings of ICHCSC 2022, Singapore: Springer Nature Singapore, 2022. 283–291 p. ISBN 978-981-19-5403-0.

MARTINS, W. C. **Algoritmo para criação de rotas de compras econômicas**. 2015. Monografia (Graduação em Engenharia de Software) - Faculdade UnB Gama, Universidade de Brasília, Brasília, 2015.

PEREIRA, B.; OLIVEIRA, P. R. Um algoritmo para calcular o menor custo de compras em supermercados considerando o preço do produto e o deslocamento. In: . [S.l.: s.n.], 2019. Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas.

SALES, H. L. B.; SANTOS, J. P. d. C. R. d. et al. **Implementação de algoritmo para cálculo de rotas IP para o RDS-Defesa**. 2019. Projeto de Fim de Curso (Curso de Graduação em Engenharia de Fortificação e Construção) - Instituto Militar de Engenharia.

SIDABUTAR, J. A.; SINULINGGA, U. The implementation of dijkstra algorithm in searching the shortest path for fire engines in medan city. **Journal of Mathematics Technology and Education**, v. 1, n. 2, p. 160–172, Mar. 2022.

SILVA, L.; ARAÚJO, F. et al. Desenvolvimento de um aplicativo mobile para navegação interna no ceulp/ulbra. In: ENCOINFO. **ENCOINFO-Congresso de Computação e Tecnologias da Informação**. [S.l.], 2020. p. 28–35.

SOUZA, F.; MAGALHãES, R.; MACHADO, W. Mobile application based on crowd computing and operational research for optimization of purchases. In: . Rio de Janeiro/RJ - Brazil: L SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 2018.

7 ANEXOS

Links para os notebooks das implementações dos códigos e casos testes, os links são de visualização, para realizar alterações, criar uma copia em seus drives.

Método Simplificado

Método Generalizado