UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUIZ HENRIQUE BIRCK VICARI

LÓGICA FUZZY E CONJUNTOS POR SIMILARIDADE APLICADOS NO DESENVOLVIMENTO DE SISTEMAS DE RECOMENDAÇÃO DE RESTAURANTES

PATO BRANCO

2023

LUIZ HENRIQUE BIRCK VICARI

LÓGICA FUZZY E CONJUNTOS POR SIMILARIDADE APLICADOS NO DESENVOLVIMENTO DE SISTEMAS DE RECOMENDAÇÃO DE RESTAURANTES

Fuzzy logic and similarity sets applied to the development of restaurant recommendation systems

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Ives Renê Venturini Pola

Coorientador: Prof. Dr. Marco Antonio de Castro Barbosa

PATO BRANCO

2023

LUIZ HENRIQUE BIRCK VICARI

LÓGICA FUZZY E CONJUNTOS POR SIMILARIDADE APLICADOS NO DESENVOLVIMENTO DE SISTEMAS DE RECOMENDAÇÃO DE RESTAURANTES

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 29 de novembro de 2023

Prof. Dr. Ives Renê Venturini Pola
Universidade Tecnológica Federal do Paraná

Prof. Dr. Dalcimar Casanova
Universidade Tecnológica Federal do Paraná

Profa. Dra. Vivane Dal Molin
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2023

Dedico este trabalho a Deus, pois desde o início Ele estava ao meu lado, e foi Ele quem tornou esses momentos possíveis.

AGRADECIMENTOS

Não poderia começar esse trabalho sem agradecer a Deus, pois foi Ele quem me deu a oportunidade, competência e capacidade para realizar tudo o que foi necessário durante o período da minha graduação.

Agradeço à minha família: mãe, pai, irmã e avós, por todo o apoio e carinho durante esse tempo.

Agradeço também à minha parceira Diulia por todo o amor e atenção que recebi durante a última metade do curso, sem você, seria bem menos agradável essa fase.

Agradeço ao Prof. Dr. Ives Renê Venturini Pola e ao Prof. Dr. Marco Antonio de Castro Barbosa pela orientação, inspiração e amizade durante a elaboração desse trabalho. Assim como agradeço aos membros da banca: Prof. Dr. Dalcimar Casanova e Profª. Drª. Kelly Lais Wiggers pelo tempo dedicado à correção da monografia.

Também, agradeço aos membros do Ganhei, se cheguei onde cheguei, com certeza foi graças ao apoio e à ajuda de todos, além de tornarem a faculdade muito mais divertida.

Finalmente, agradeço a todos que estiveram comigo durante a graduação e que contribuíram nessa fase da minha vida.

Simplesmente não posso ir sem saber para onde.
(Bilbo Bolseiro, O Hobbit, Uma Jornada Inesperada).

RESUMO

Sistemas de recomendação têm grande importância na atualidade, sendo utilizados em diversas aplicações, como *e-commerce* e redes sociais. Este trabalho apresenta uma abordagem de sistema de recomendação fundamentada em lógica *fuzzy*, roteamento e conjuntos por similaridade. Para o desenvolvimento da pesquisa, foi proposta uma forma de seleção de elementos utilizando lógica *fuzzy*, utilizando os conceitos de *fuzzyfication interface* e intersecção em conjuntos *fuzzy*, considerando as características de um elemento e a relação de distância geográfica entre cada elemento e os outros pontos de uma rota. Além disso, com base em um elemento, é buscado encontrar outros elementos similares a esse dentro de um conjunto por similaridade. O sistema desenvolvido nesse trabalho foi testado, validado comparando o tamanho da rota encontrada com uma rota gerada pelo algoritmo guloso e comparando o tempo da busca dentro do conjunto por similaridade gerado com o tempo de busca por método *Reverse K Nearest Neighbors* (RKNN).

Palavras-chave: lógica *fuzzy*; conjuntos por similaridade; sistemas de recomendação; problema do caixeiro viajante; np-completude.

ABSTRACT

Recommendation systems are of great importance today and are used in various applications such as e-commerce and social networks. This work presents a recommendation system approach based on fuzzy logic, routing and similarity sets. To develop the research, a way of selecting elements using fuzzy logic is proposed, using the concepts of fuzzyfication interface and intersection in fuzzy sets, considering the characteristics of an element and the relationship of geographical distance between each element and the other points on a route, in addition, based on an element, it is sought to find other elements similar to it within a set by similarity. The system developed in this work is tested and validated by comparing the size of the route found with a route generated by the greedy algorithm and by comparing the search time within the similarity set generated with the search time using the RKNN method.

Keywords: fuzzy logic; similarity sets; recommender system; travelling salesman problem; np-completeness.

LISTA DE FIGURAS

Figura 1 – Algoritmo 2-opt **23**
Figura 2 – Busca por distância **24**
Figura 3 – Fluxograma de metodologia de pesquisa **29**
Figura 4 – Comparação de quantidade entre categoria restaurante e outras categorias **34**
Figura 5 – Comparação de quantidade entre categorias sem a categoria "Restaurante" **34**
Figura 6 – Localização das instâncias **35**
Figura 7 – Correlação entre tipos de restaurantes **36**
Figura 8 – Primeira etapa da construção do *SimSet* **39**
Figura 9 – Segunda etapa da construção do *SimSet* **39**

LISTA DE TABELAS

Tabela 1 – Graus de pertencimento de objetos no conjunto de objetos parecidos com um carro **17**
Tabela 2 – Número de dados encontrados para cada tipo de estabelecimento . . . **33**
Tabela 3 – Exemplo de tipos de restaurante para restaurante fictício **36**
Tabela 4 – Comparação entre algoritmo *Greedy Randomized Adaptive Search Procedure* (GRASP) e algoritmo guloso para a primeira instância **37**
Tabela 5 – Descrição das instâncias **38**
Tabela 6 – Comparação de tempo de execução entre *SimSet* e RKNN **40**

LISTAGEM DE CÓDIGOS FONTE

Listagem 1 – Algoritmo guloso para resolver o Problema do caixeiro viajante (PCV) 21
Listagem 2 – Método GRASP generalizado 22
Listagem 3 – Algoritmo para reuqisição de dados através da API do *Google Maps* 47
Listagem 4 – Algoritmo de Seleção de Restaurantes com lógica *fuzzy* 49
Listagem 5 – Etapa construtiva do método GRASP para PCV 52
Listagem 6 – Algoritmo 2-opt 55
Listagem 7 – Algoritmo guloso para resolver o PCV 57
Listagem 8 – Algoritmo para a construção de *SymSets* baseado em rkInn 59
Listagem 9 – Algoritmo de busca por RKNN 62

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	*Application Programming Interface*
GRASP	*Greedy Randomized Adaptive Search Procedure*
KNN	*K Nearest Neighbors*
NN	*Nearest Neighbour*
NYC	*New York City*
PCV	Problema do caixeiro viajante
RKNN	*Reverse K Nearest Neighbors*
RNN	*Reverse Nearest Neighbour*
SGBDR	Sistema gerenciador de banco de dados relacional
SO	Sistema operacional

SUMÁRIO

1 INTRODUÇÃO 13

1.1 Objetivos 14

1.1.1 Objetivo Geral 14

1.1.2 Objetivos Específicos 14

1.2 Estrutura do Trabalho 15

2 REFERENCIAL TEÓRICO 16

2.1 Sistemas de Recomendação 16

2.2 Lógica *Fuzzy* 17

2.2.1 Operações em Conjuntos *Fuzzy* 18

2.3 Problemas NP-Completos 19

2.3.1 Problema do Caixeiro Viajante 20

2.3.2 Algoritmo Guloso 21

2.3.3 Método GRASP 21

2.3.4 Algoritmo 2-opt 23

2.4 Buscas por Similaridade 23

2.4.1 *Reverse K Nearest Neighbours* 25

2.4.2 Conjuntos por Similaridade 25

2.5 Trabalhos Relacionados 26

2.5.1 Lógica *Fuzzy* 26

2.5.2 Similaridade 27

3 MATERIAIS E MÉTODOS 28

3.1 Materiais 28

3.2 Métodos 28

3.2.1 Recomendação por Lógica *Fuzzy* e Trajetos 30

3.2.2 Recomendação por Conjuntos de Similaridade 31

4 RESULTADOS 32

4.1 Escopo do Sistema 32

4.2 Resultados Obtidos 33

4.2.1 Extração dos Dados 33

4.2.2 Mineração de Relações *Fuzzy* e Seleção de Elementos 35

4.2.2.0.1 *Seleção de Dados* 37
4.2.3 Busca e Otimização da Rota 37
4.2.4 Construção e Uso de Conjuntos por Similaridade 38
5 CONCLUSÃO 41
REFERÊNCIAS 42

APÊNDICES 45
APÊNDICE A – ALGORITMO PARA REQUISIÇÃO DE DADOS 47
APÊNDICE B – CÓDIGO PARA MINERAÇÃO DE RELAÇÕES *FUZZY* . . 49
APÊNDICE C – ALGORITMO GRASP 52
APÊNDICE D – ALGORITMO 2-OPT 55
APÊNDICE E – ALGORITMO GULOSO 57
APÊNDICE F – CONSTRUÇÃO DO *SIMSET* 59
APÊNDICE G – ALGORITMO *REVERSE K NEAREST NEIGHBOURS* . . 62

1 INTRODUÇÃO

Sistemas de recomendação tem grande participação em diferentes setores. Esse tipo de sistema busca encontrar a melhor ou as melhores opções para um usuário, baseado em regras escolhidas pela aplicação. Por exemplo, as escolhas passadas ou informações de usuários com um perfil parecido podem ser consideradas. De acordo com Liu (2014), sistemas de recomendação são ferramentas baseadas em *software* que fazem a tomada de decisão e respondem com uma sugestão para o usuário. O autor também destaca que diversos sites, como *Amazon* e *Google News* utilizam sistemas de recomendação complexos visando encontrar itens melhores para os usuários.

Serviços como o *Google Maps* ou *TripAdvisor* também usam sistemas de recomendação para encontrar estabelecimentos próximos ao usuário que possam satisfazê-lo. Por exemplo, ao pesquisar no serviço um restaurante, o sistema retorna diversas opções desse tipo de estabelecimento. As relações encontradas nesses sistemas são, muitas vezes, booleanas, e essa representação pode não ser a mais adequada para os conjuntos. Além disso, os estabelecimentos podem ser similares entre si, e um deles pode ter suas características representadas por outros.

Contudo, as relações entre estabelecimentos são complexas, visto que existem vários tipos de restaurantes, que podem ou não ser semelhantes entre si. Além disso, alguns sistemas de recomendação devem levar em conta outros fatores, como outros conjuntos de itens da escolha de um usuário. Por exemplo, um sistema de recomendação de restaurantes pode considerar os locais em que a pessoa que vai utilizá-lo pretende visitar, assim expandindo o conjunto de restaurantes que podem ser recomendados, em comparação com o caso em que apenas um ponto é utilizado para avaliar proximidade.

Tendo em vista o conjunto de pontos e as relações complexas entre os estabelecimentos, os problemas envolvem a criação de rotas e a sugestão de itens com base no conhecimento prévio dos outros pontos para que seja possível utilizar a proximidade como critério de escolha. O problema de criar rotas pode ser reduzido ao problema do Caixeiro Viajante, problema NP-Completo, ou seja, de acordo com Garey e Johnson (1979), não é possível garantir a solução exata em tempo polinomial. Tendo em vista a complexidade desse problema, busca-se resolvê-lo com uma meta-heurística que encontre soluções aceitáveis sem explorar todo o conjunto de possíveis respostas, (GENDREAU; POTVIN, 2010).

Ainda, para explorar as relações entre tipos de itens para a sugestão, em que um item pode ser semelhante a outro, propõe-se a utilização de lógica *fuzzy*, essa teoria define conjuntos que se diferem dos conjuntos clássicos ao introduzir que um elemento contém um grau de participação no conjunto, não necessariamente apenas pertencendo ou não a este. Este trabalho buscou aplicar a lógica *fuzzy* e as regras definidas por Zadeh (1965) para modelar os elementos e verificar as melhores respostas. Além disso, foi avaliado o uso de conjuntos por similaridade,confirme definido por Pola *et al.* (2015), que sintetizam conjuntos de dados com-

plexos, escolhendo os itens que melhor representam partes do conjunto, para encontrar itens que se assemelham às melhores escolhas, ampliando assim capacidade de recomendação.

No trabalho de Silva *et al.* (2022), o autor propõe uma forma de mineração de relações geográficas em objetos com localização desconhecida, possibilitando o roteamento entre pontos os quais a localização não é perfeitamente definida. Esse trabalho se assemelha ao que será apresentado, porém, ele não resolve o problema de escolha de itens, o qual este trabalho propõe-se a resolver, explorando, em vez de localização desconhecida, relações *fuzzy* entre as características dos itens. Ademais, Chen (1995) propõe uma forma de, utilizando lógica *fuzzy*, medir a similaridade entre conjuntos vagos diferentes, enquanto Xu e Chen (2008) propõe um estudo acerca de medidas de distância em conjuntos *fuzzy*.

Para estudar os resultados e avaliar a solução proposta, foi coletado e utilizado um conjunto de dados contendo estabelecimentos que trabalham com alimentos, tais quais restaurantes e cafeterias, hospedagens e pontos turísticos da cidade de *New York City* (NYC), (Estados Unidos da América), por conta da maior quantidade de dados para estudo. A partir desse conjunto e uma entrada de usuário, será buscado recomendar qual o melhor restaurante e quais outros são similares à melhor escolha.

Portanto, a solução apresentada nesse trabalho combina métodos de solução para os problemas de roteamento, utilizando critérios de decisão baseados em lógica *fuzzy* na escolha de itens, assim como conjuntos por similaridade para agrupar os itens. Dessa forma, essa pesquisa contribui no avanço de soluções de otimização e exploração combinatória, no entendimento e uso de dados difusos e na modelagem de sistemas de recomendação.

1.1 Objetivos

A seguir, são apresentados o objetivo geral e os objetivos específicos desenvolvidos nessa pesquisa.

1.1.1 Objetivo Geral

Esse trabalho teve como objetivo desenvolver um sistema de recomendação, utilizando métodos computacionais que consideram as características *fuzzy* e a similaridade entre itens, levando em conta a localização de outros itens relacionados.

1.1.2 Objetivos Específicos

- Coletar os dados de estabelecimentos gastronômicos, hospedagens e pontos turísticos.

- Criar uma métrica dentro das características dos itens que difere das métricas *fuzzy* no sistema, levando em conta os critérios de usuários quando ao custo e qualidade.
- Integrar a métrica externa na decisão de escolha dos itens, que consideram a proximidade aos outros itens.
- Utilizar uma meta heurística para encontrar a rota que possua uma distância menor que o algoritmo guloso.
- Construir um conjunto por similaridade com os itens que devem ser escolhidos e utilizá-lo para expandir o conjunto de sugestões.

1.2 Estrutura do Trabalho

Este trabalho está dividido da seguinte forma:

- O Capítulo 2 apresenta os conceitos mais importantes no desenvolvimento da pesquisa, são eles: sistemas de recomendação na seção 2.1, lógica *fuzzy*, conceitos e manipulação de conjuntos na seção 2.2, problemas NP-completos, suas características e estratégias para resolvê-los na seção 2.3, e os conceitos e formas de buscas por similaridade na seção 2.4. Ainda, na seção 2.5 é feita a comparação entre outros trabalhos que envolvem as áreas de pesquisa abordadas em comum com esse.
- O Capítulo 3 apresenta as especificações de *hardware* e *software* para o desenvolvimento da pesquisa e a metodologia e fluxo de trabalho para desenvolver o projeto.
- No Capítulo 4 os resultados obtidos para validar as etapas da metodologia são apresentados e discutidos.
- O Capítulo 5 conclui o trabalho, apresentando resumidamente o que foi elaborado e qual interpretação pode ser feita com o estudo dessa pesquisa.

Além disso, os apêndices apresentam alguns dos algoritmos desenvolvidos ou utilizados durante a pesquisa.

2 REFERENCIAL TEÓRICO

Nesse capítulo, são apresentados e discutidos os conceitos teóricos relevantes para a pesquisa realizada neste trabalho, tais como: sistemas de recomendação, lógica *fuzzy*, problemas NP-completos e buscas por similaridade.

2.1 Sistemas de Recomendação

De acordo com Ricci *et al.* (2011), um sistema de recomendação tem como objetivo encontrar itens (aquilo que o sistema busca recomendar) que mais se adéquem para um usuário, utilizando processos de decisão via *software*, que podem ser personalizáveis ou não. Assim, Ricci *et al.* (2011) apresenta como exemplos para cada uma como:

- Sistema de recomendação personalizável: pode ser por usuário ou grupos de usuários, por exemplo o site “Amazon.com” personaliza a sua loja online para cada cliente.
- Sistema de recomendação não personalizável: são mais simples, como recomendar um mesmo CD para todas as pessoas por ele ter sido o mais vendido no ano. Esse tipo de sistema de recomendação, contudo, geralmente não é alvo de pesquisas no ramo de sistemas de recomendação.

> Na sua forma mais simples, sistemas de recomendação personalizáveis são listas classificadas de itens. Ao fazer essa classificação, o sistema de recomendação tenta prever qual produto ou serviço é mais adequado, baseado nas preferências e restrições do usuário. (RICCI *et al.*, 2011, 2).

Além disso, Liu (2014) apresenta a recomendação baseada em usuários de vizinho mais próximo, em que, para avaliar as recomendações de um usuário atual, são analisadas as ações de outros usuários parecidos com ele no passado, baseando-se na suposição de que as ações dos usuários mantém-se consistentes durante o tempo. Ainda, ele apresenta a recomendação baseada em conteúdo, na qual é necessário obter a descrição das características do item e o perfil ou interesses do usuário e, com base nessas informações, encontrar qual ou quais itens melhor atendem o usuário analisado.

Assim, o objetivo de um sistema de recomendação é encontrar um item que seja a melhor ou uma das melhores opções para o usuário, tendo como entrada informações dadas por ele. Ainda, algumas razões pelas quais um serviço pode se beneficiar de sistemas de recomendação oferecidos para o usuário são: aumento no número de vendas, vendas mais diversas, mais satisfação do usuário, maior fidelidade por parte dos clientes e melhor compreensão do que o usuário deseja (RICCI *et al.*, 2011).

2.2 Lógica *Fuzzy*

De acordo com Lipschutz e Lipson (2013), a teoria dos conjuntos é de suma importância na matemática, na qual conjuntos são coleções de objetos bem definidas. Além disso, segundo Menezes (2013), na teoria dos conjuntos clássica, os elementos podem apenas pertencer ao conjunto (denotado como $a \in A$ ou a pertence ao conjunto A) ou não pertencer ($a \notin A$). Todavia, de acordo com Zadeh (1965), essa teoria consegue representar muito bem diversas situações e conjuntos, porém ela pode não ser tão adequada para outras, por exemplo para definir um conjunto de objetos parecidos com carros, nesse caso, um carro convencional pertenceria a tal conjunto, mas uma motocicleta pode ser considerada como um objeto parecido com um carro, levando ao problema "a motocicleta deve ser inserida no conjunto?". Dessa forma, o conjunto mencionado anteriormente pode ser reconstruído sem necessariamente usar as regras clássicas dos conjuntos.

Para resolver problemas como o descrito anteriormente, Zadeh (1965) definiu a teoria de conjuntos *fuzzy*. O autor propõe uma nova abordagem aos conjuntos na qual, em vez de um elemento pertencer ou não ao conjunto, ele possui um grau contínuo de pertencimento, que varia entre 0 (zero) e 1 (um), ou seja, usando como exemplo do conjunto de objetos parecidos com um carro, os seguintes objetos poderiam ter os graus de pertencimento apresentados na Tabela 1.

Tabela 1 – Graus de pertencimento de objetos no conjunto de objetos parecidos com um carro

Objeto	Grau de pertencimento
Carro	1.0
Motocicleta	0.7
Porta	0.3
Animal	0.0

Fonte: Autoria própria (2023).

Na Tabela 1, é possível perceber que o objeto com maior grau de pertencimento é o próprio carro, afinal, ele representa o elemento que o conjunto busca a semelhança, em seguida, é a motocicleta, pois ela tem algumas características em comum com o carro, como motor, função (meio de transporte), entre outros; a porta possui um grau mais baixo, mas ainda assim diferente de 0 (zero), pois um carro tem uma porta, então os objetos tem características em comum. Por fim, o animal possui grau de pertencimento 0 (zero) pois ele compartilha poucas ou nenhuma característica com carros. É importante ressaltar que o exemplo da Tabela 1 é teórico e os graus de pertencimento podem variar de acordo com as especificações do conjunto.

De uma maneira formal, um conjunto *fuzzy* pode ser definido como:

> Com X sendo um espaço de objetos, e um elemento qualquer de X sendo denotado por x. Assim: $X = x$.
>
> Um conjunto *fuzzy* A em X é caracterizado por uma função de pertencimento $f_A(x)$, que associa cada ponto em X a um número real

> no intervalo $[0, 1]$, com o valor de $f_A(x)$ em x representando o "grau de pertencimento" de x em A. Assim, quanto mais próximo de 1 for o valor de $f_A(x)$, maior o pertencimento de x em A.(ZADEH, 1965, p.2).

Dessa forma, ainda segundo Zadeh (1965), um conjunto *fuzzy* no qual os resultados de $f_A(x_i)$ para cada x_i podem assumir apenas os valores 0 ou 1 tem o mesmo efeito que um conjunto clássico.

Assim, é necessário entender como construir e manipular um conjunto *fuzzy*, portanto, Lee (1990) define os principais componentes para um sistema de controle *fuzzy*, são eles: *Fuzzyfication interface*, base de conhecimento, lógica de tomada de decisão e *Defuzzyfication interface*. As funções de cada componente são explicadas a seguir.

- *Fuzzyfication interface*: esse componente deve medir os valores de variáveis de entrada e transformá-las para os valores *fuzzy* do universo de discussão.
- Base de conhecimento: provê as definições necessárias para fazer o controle e manipulação dos dados no sistema de controle.
- Lógica de tomada de decisões: simula a decisão humana baseado nos conceitos *fuzzy* definidos para a aplicação.
- *Defuzzyfication interface*: converte as variáveis de saída novamente para o conjunto o estado inicial - faz o contrário da *Fuzzyfication interface*.

Tais componentes foram definidos para sistemas de controle voltados para aplicações em engenharia elétrica, portanto, para usos em outras aplicações, elas podem ser adaptados. Além disso, assim como nos conjuntos clássicos, os conjuntos *fuzzy* tem operações como união e interseção definidas, as quais são apresentadas na subseção 2.2.1.

2.2.1 Operações em Conjuntos *Fuzzy*

Para manipular conjuntos *fuzzy*, Zadeh (1965) define as operações de igualdade, união e interseção para esses conjuntos. Para as operações a seguir, consideram-se dois conjuntos *fuzzy* A e B, com funções de pertencimento $f_A(x)$ e $f_B(x)$ respectivamente. As regras definidas por Zadeh (1965) são:

- Igualdade: dois conjuntos A e B são iguais se a Equação 1 for satisfeita.

$$f_A(x) = f_B(x), \forall x \in X \tag{1}$$

  Ou seja, todos os elemento tem o mesmo grau de pertencimento em ambos os conjuntos.

- União: a união entre os conjuntos A e B é dada por um conjunto C, o qual a função de pertencimento $f_C(x)$ é apresentada na Equação 2.

$$f_C(x) = Max[f_A(x), f_B(x)], x \in X \tag{2}$$

  Assim, o grau pertencimento de um elemento em C é dado pelo maior grau de pertencimento entre os conjuntos A e B.

- Interseção: a interseção entre os conjuntos A e B é dada por um conjunto C, o qual a função de pertencimento $f_C(x)$ é apresentada na Equação 3:

$$f_C(x) = Min[f_A(x), f_B(x)], x \in X \tag{3}$$

  Portanto, o grau de pertencimento de um elemento em C é dado pelo menor grau de pertencimento entre os conjuntos A e B.

Com essas regras definidas, ainda é possível definir as operações *AND* e *OR* lógicas, de forma que:

- *AND* lógico: para dois elementos x e y, a operação lógica *AND* resulta no menor grau de pertencimento entre os elementos (equivalente à Equação 3).

- *OR* lógico: para dois elementos x e y, a operação lógica *AND* resulta no maior grau de pertencimento entre os elementos (equivalente à Equação 2).

Por fim, a validade dessas operações pode ser verificada ao aplicá-las em conjuntos *fuzzy* onde os graus de pertencimento são dados apenas por 0 ou 1 e comparando o resultado com o equivalente desses nos conjuntos clássicos.

2.3 Problemas NP-Completos

Garey e Johnson (1979) definem duas classes de problemas, os problemas P e os problemas NP, segundo os autores, um problema P é aquele que pode ser resolvido por algoritmos em tempo polinomial, enquanto os problemas NP-completos só podem ser verificados de forma exata com tempo exponencial, nos quais o objetivo é verificar se existe uma solução melhor que um limite definido (por exemplo: se é possível encontrar uma rota com distância menor que d). Além disso, ou autores também discorrem sobre os problemas NP-difíceis, que se assemelham aos problemas NP-completos, porém o objetivo é encontrar a melhor resposta, assim, faz-se necessário verificar todo o conjunto de soluções possíveis. Ainda, de acordo com Britannica (2023), para resolver problemas NP-completos, geralmente é utilizado um algoritmo de tempo polinomial que aproxima o resultado ótimo do problema, encontrando uma solução boa, mas

não necessariamente a melhor global. Ademais, um dos exemplos de problemas NP-completos é o PCV, que é também um problema de roteamento, que pode ser generalizado pelo circuito hamiltoniano (GAREY; JOHNSON, 1979, 56).

O problema do circuito hamiltoniano pode ser definido como:

> Considerando uma instância aleatória de cobertura de vértices dada pelo grafo $G = (V, E)$ e um inteiro positivo $K \leq |V|$. É possível construir um grafo $G' = (V', E')$ no qual G' tem um circuito hamiltoniano se, e apenas se, G tem uma cobertura de vértices de tamanho K ou menor. (GAREY; JOHNSON, 1979, 56).

Por definição, de acordo com Filho (2016, 4): "dado um grafo não direcionado $G = (V, E)$, uma cobertura por vértices é um subconjunto de vértices S tal que, para cada aresta $(u, v) \in E, u \in S$, ou $v \in S$ (ou ambos). O tamanho da cobertura por vértices é o número de vértices em S". Ou seja, a cobertura de vértices é um conjunto de vértices dentro de um grafo no qual todas as arestas estão conectadas a pelo menos um dos vértices desse conjunto.

A seguir, será discorrido sobre o PCV e suas soluções.

2.3.1 Problema do Caixeiro Viajante

Garey e Johnson (1979) definem o PCV como um problema no qual há um conjunto de cidades, na qual é buscado encontrar uma rota com distância menor que uma distância limite B, a qual passe por todas as cidades apenas uma vez e volte para a cidade de origem. De maneira mais formal:

> É tido como instância: um conjunto finito $C = c_1, c_2, ..., c_m$, uma "distância" $d(c_i, c_j) \in Z^+$ para cada par de cidades $c_i, c_j \in C$ e um limite $B \in Z^+$, onde Z^+ denota o conjunto dos números inteiros positivos. É buscado se há uma rota para todas essas cidades em C tendo distância total B, ou seja, uma ordem $< c_{\pi(1)}, c_{\pi(2)}, ..., c_{\pi(m)} >$ de C que:
>
> $$[\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)})] + d(c_{\pi(m)}, c_{\pi(1)})$$
>
> (GAREY; JOHNSON, 1979, 19).

Ainda, de acordo com Garey e Johnson (1979), é possível transformar esse problema de decisão (que busca responder sim ou não para a distância menor que B) em um problema de maximização ou minimização. Por fim, o problema do caixeiro viajante pode ser generalizado e utilizado para resolver problemas de roteamento que visam encontrar a menor distância possível.

2.3.2 Algoritmo Guloso

De acordo com Black (2005) uma estratégia de solução gulosa é “um algoritmo que sempre escolhe a solução com melhor resultado imediato, ou melhor local, enquanto procura a resposta”. Ou seja, o algoritmo guloso sempre busca encontrar a resposta que, para o momento de execução atual, parece ser mais vantajosa, mesmo que possa vir a não produzir um bom resultado no futuro. O autor cita vantagens e desvantagens de estratégias de solução gulosa: a grande vantagem é ser, geralmente, mais rápido em tempo de execução; a desvantagem é que, tendo como exemplo o circuito hamiltoniano, mencionado anteriormente, o algoritmo pode não encontrar o menor caminho.

O algoritmo guloso para resolver o PCV, de acordo com Kim, Shim e Zhang (1998), funciona como mostrado na Listagem 1.

1. Começar no primeiro nó.
2. Enquanto houver nós não visitados, faça:
    - Calcular a distância para todos os nós não visitados.
    - Viajar para o nó cuja distância para o atual seja a menor dentre os outros.
    - Marcar o nó para o qual foi viajado na rota e marcar como visitado.
3. Viajar novamente para o nó inicial

Listagem 1 – Algoritmo guloso para resolver o PCV

Para Kim, Shim e Zhang (1998), essa é a solução mais simples para resolver o PCV.

Devido a natureza de funcionamento do algoritmo guloso, Gutin, Yeo e Zverovich (2002) afirma que ele não possui bons resultados para as variações simétricas (onde as distâncias entre nós são as mesmas em ambas as direções) e assimétricas (a distancia de um nó para outro pode variar dependendo da direção) do PCV, encontrando soluções que se distanciam muito da distância ótima. Portanto, é necessário outro método para resolver o PCV, assim, o método GRASP, que contorna os defeitos do algoritmo guloso, será apresentado a seguir na subseção 2.3.3.

2.3.3 Método GRASP

De acordo com Resende e Ribeiro (2010), o GRASP é uma meta-heurística (uma estratégia de busca para analisar as possíveis soluções de um problema sem analisar todas as possibilidades) para resolver problemas de otimização combinatória, que consiste em encontrar uma solução e, a partir dessa, investigar soluções vizinhas, buscando encontrar um ótimo local. A cada iteração de um algoritmo baseado no método GRASP, uma nova solução pode

ser encontrada, assim, podem ser encontrados novos ótimos locais, os quais podem chegar em ótimos globais.

O método GRASP para resolver problemas de minimização pode ser definido como na Listagem 2.

Entrada: Conjunto de elementos C, número máximo de iterações N.
Saída: Solução
Início:

1. Enquanto o número de iterações for menor que N, faça:
    a) Encontre a solução da etapa de construção.
    b) Se a solução não for viável:
        i. Repare a solução.
    c) Execute a busca local na vizinhança da solução.
    d) Atualize a solução se ela for melhor que a melhor solução atual.
2. Retorne a melhor solução.

Fim.

Listagem 2 – Método GRASP generalizado

Uma solução não viável é aquela que não respeita alguma das restrições do problema. De acordo com Resende e Ribeiro (2010), na etapa de construção, ela deve avaliar os elementos (por exemplo para o PCV, deve se avaliar a distância entre as cidades ou vértices), de forma a encontrar uma lista restrita de elementos, com as melhores possibilidades (no caso do PCV pode se utilizar os vértices cuja distância para o atual é até 30% maior que a melhor distância de qualquer vértice disponível até o atual), assim, será escolhido um elemento dessa lista de forma aleatória. Dessa forma, o algoritmo encontra novas soluções, visto que para cada execução ele pode mudar a vizinhança das soluções e tem mais chances de encontrar o ótimo global.

Como a resposta do GRASP não é necessariamente um ótimo local, apenas uma região desta vizinhança, é necessário executar uma etapa de busca local em seguida, que visa encontrar o ótimo local para a vizinhança encontrada na etapa de construção. Um exemplo de busca local é o algoritmo 2-opt, que será apresentado na subseção 2.3.4.

É importante destacar que o algoritmo apresentado nesse capítulo é apenas uma generalização e as etapas de construção (principalmente na criação da lista restrita) e busca local devem ser definidos diferentemente para cada propósito de aplicação. Uma versão da etapa de construção do GRASP para resolver o PCV é apresentada no Apêndice C.

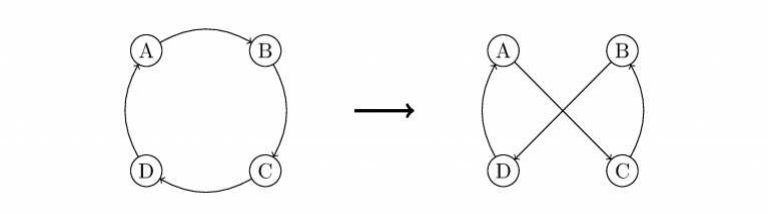
A seguir, será apresentado na subseção 2.3.4 o algoritmo 2-opt, uma opção para satisfazer a necessidade de busca local do método GRASP canônico.

2.3.4 Algoritmo 2-opt

O algoritmo 2-opt é uma estratégia de busca local que necessita de uma solução já construída, e tem como objetivo melhorá-lá. De acordo com Johnson e McGeoch (2008), o algoritmo pode ser considerado como uma forma de busca nas vizinhanças da solução original, que pode ser encontrada alterando apenas um movimento (no caso de problemas de roteamento) e prosseguir com essa operação iterativamente.

Considerando uma rota já construída, Johnson e McGeoch (2008) definem o funcionamento do algoritmo 2-opt em eliminar duas arestas que ligam os vértices na rota, de forma que a rota esteja separada em duas partes, em seguida, essas partes são ligadas de forma diferente da original e é avaliado se essa rota é melhor ou pior que a anterior. De forma similar, o algoritmo 3-opt executa o mesmo procedimento, porém considerando trocas entre 3 arestas.

Figura 1 – Algoritmo 2-opt



Fonte: (GAZDA, 2019).

Pela Figura 1, é possível visualizar o mecanismo do algoritmo 2-opt, o qual consiste em trocar as arestas de um grafo direcionado, que representa a rota, caso encontre uma rota melhor. O algoritmo procede testando trocas de arestas entre todos os nós, e se encontrar uma troca que seja benéfica (ou seja, diminua a distância, para o PCV), a troca de arestas é confirmada e a direção de todas as arestas entre as duas que foram trocadas é invertida.

Além disso, de acordo com Rocki e Suda (2012), utilizando uma série de execuções do algoritmo 2-opt com recomeços aleatórios, é possível encontrar a melhor solução global, já que o fator probabilístico envolvido no recomeço do algoritmo permite que a busca escape de melhores locais, contudo, esse processo exigiria tempo de execução infinito.

Por fim, Kim, Shim e Zhang (1998) comparam alguns algoritmos para resolver o PCV, e demonstram que, em comparação com o algoritmo 3-opt, o 2-opt é mais rápido e consegue encontrar resultados melhores em todas as instâncias de teste em que o algoritmo 3-opt pôde ser executado, em função da maior complexidade da função. Portanto, é possível perceber que entre os algoritmos k-opt, a estratégia com $k = 2$ é uma boa escolha.

2.4 Buscas por Similaridade

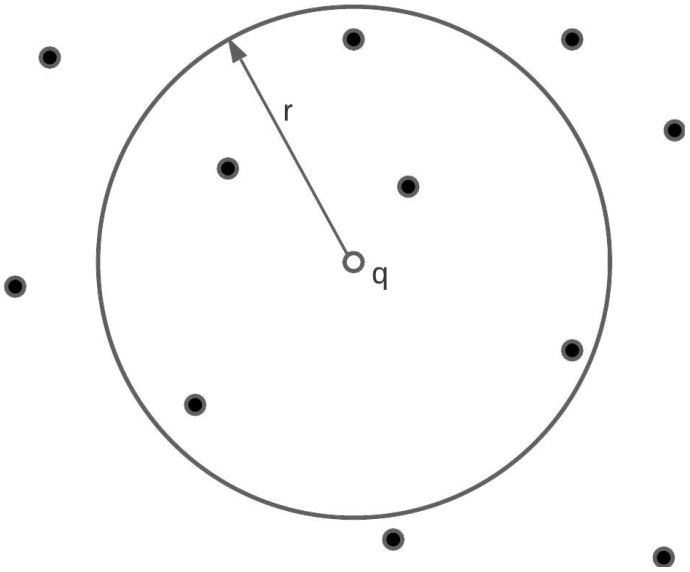
Segundo Zezula *et al.* (2005), os dados digitais modernos consistem cada vez mais de vetores com muitas dimensões, nos quais as formas de buscas convencionais não são muito

eficientes, dessa forma, é melhor utilizar abordagens que se baseiem em proximidade, tais como a similaridade. Ainda para Zezula *et al.* (2005), o interesse da comunidade da área de bancos de dados tem crescido graças a necessidade de trabalhar com grandes volumes de dados, assim,o desempenho de buscas por similaridade é um assunto de importante discussão.

> Uma busca por similaridade é definida explicitamente ou implicitamente por um objeto de busca q e uma restrição na forma de proximidade desejada, tipicamente expressa como uma medida de distância. (ZEZULA *et al.*, 2005, 15).

Zezula *et al.* (2005) atesta que a forma mais comum para as buscas por similaridade consiste em utilizar uma faixa de distância, onde é especificado um objeto e uma distância, e qualquer elemento que esteja mais próximo que essa distância faz parte da resposta. O funcionamento da busca por distância pode ser visto na Figura 2.

Figura 2 – Busca por distância



Fonte: Adaptado de (ZEZULA *et al.*, 2005, 16).

Na Figura 2, q é o elemento de referência e r é a distância limite, dessa forma, todos os elementos que estejam dentro do círculo de raio r com origem em q fazem parte da resposta da busca por distância. Assim, é possível perceber que com r grande o suficiente, todos os elementos serão retornados e com r pequeno o suficiente, o conjunto de resposta será vazio (considerando que não há outros elementos idênticos à q).

Zezula *et al.* (2005) ainda define outras formas importantes de buscas por similaridade, são elas: *Nearest Neighbour* (NN), *Reverse Nearest Neighbour* (RNN) e junção por similaridade, as quais podem ser combinadas entre si. A seguir, na subseção 2.4.2 e na subseção 2.4.1, serão apresentadas duas formas de busca por similaridade, as quais serão futuramente comparadas em relação ao tempo de busca.

2.4.1 *Reverse K Nearest Neighbours*

De acordo com Papadias e Tao (2009), uma busca reversa pelo vizinho mais próximos pode ser definida como uma busca em um espaço dimensional que retorna todos os elementos que tem o elemento de comparação como seu vizinhos mais próximo, ou, mais formalmente:

> Dado um conjunto de dados multidimensional P e um ponto q, um RNN recupera todos os pontos $p \in P$ que tem q como seu vizinhos mais próximo. O conjunto $RNN(q)$ dos vizinhos mais próximos reversos de q é chamado de conjunto de influência de q. (PAPADIAS; TAO, 2009).

Dessa forma, o conceito de RNN pode ser expandido para RKNN, que recupera os dados que possuem um elemento q como um de seus k vizinhos mais próximos, sendo $k \exists Z, k > 0$, (TAO *et al.*, 2007). De acordo com Wu *et al.* (2008), essa estratégia de busca é utilizada para encontrar conjuntos de influência em dados, já que a distância entre os elementos pode indicar a influência de um sobre o outro, sendo que, quanto menor a distância, maior a influência desses elementos.

Ainda, segundo Zezula *et al.* (2005), a distância entre os objetos não é a única variável que importa, visto que objetos muito próximos podem não ser parte da resposta do RKNN, enquanto objetos distantes podem estar na resposta do algoritmo.

Por fim, Singh, Ferhatosmanoglu e Tosun (2003) descreve um método para encontrar o vizinho mais próximo reverso de forma exata, utilizando força bruta, que consiste da seguinte forma: “Primeiramente, encontrar o vizinho mais próximo, de cada ponto e determinar quais pontos tem tem o ponto inicial como seu vizinho mais próximo.”(SINGH; FERHATOSMANOGLU; TOSUN, 2003).

Contudo, ainda de acordo com Singh, Ferhatosmanoglu e Tosun (2003) esse algoritmo tem ordem de $O(n^2)$ para a sua complexidade, o que, em grandes conjuntos, pode ser uma tarefa muito demorada e não se tornar viável. Esse mesmo problema pode ser aplicado para o RKNN, ou seja, se o elemento deve ser um dos k vizinhos mais próximos (onde k é um número inteiro maior que 0), em vez do vizinho mais próximo.

2.4.2 Conjuntos por Similaridade

A teoria dos conjuntos tradicional não permite que haja dois elementos iguais dentro de um mesmo conjunto, porém, como mencionado anteriormente, grande parte dos dados atuais é multidimensional e dificilmente podem ser comparadas por igualdade, dessa forma, é necessária uma nova abordagem de conjuntos para guardar esses novos dados. Pola *et al.* (2015) define os conjuntos por similaridade e *SimSets*, uma forma de armazenar dados que não são exatamente iguais, mas muito parecidos.

Como definido por Pola *et al.* (2015), um conjunto por similaridade consiste em utilizar uma métrica de distância, chamada de distância de similaridade suficiente (ξ), e analisar os dados como um espaço métrico $M = \langle \mathbb{S}, d \rangle$ (no qual $\mathbb{S}$ é o conjunto de elementos e d a métrica de distância), de forma que dois elementos $s_n, s_m \in \mathbb{S}$ que possuam $d(s_n, s_m) \leq \xi$ sejam considerados como o mesmo, dentro do propósito da aplicação. Com base em ξ, $\mathbb{S}$ e d definidos, é possível construir um conjunto por similaridade, no qual não há elementos cujo $d(s_m, s_n) \leq \xi$ - é importante destacar que variar a distância limite ξ pode alterar o conjunto resultante. De acordo com Pola *et al.* (2015), o objetivo de um conjunto por similaridade é evitar que elementos que sejam muito parecidos possam interferir nas consultas.

Para extrair um conjunto por similaridade, Pola *et al.* (2015) propõe o algoritmo *Distinct*, um algoritmo não determinístico que consiste em gerar um grafo de similaridade (um grafo não direcionado com arestas em elementos cujo $d(s_m, s_n) \leq \xi$) e extrair o conjunto por similaridade a partir do grafo. Esse algoritmo possui duas políticas:

- *Max*: busca alcançar o conjunto por similaridade com o maior conjunto de elementos, construindo-os com os vértices de menor grau do grafo de similaridade.
- *min*: constrói o conjunto com o menor número possível de elementos, utilizando para isso os nós com maior grau no grafo.

2.5 Trabalhos Relacionados

Esse trabalho envolve duas áreas de pesquisa diferentes: lógica *fuzzy*, e buscas e conjuntos por similaridade, portanto, essa seção está separada entre os trabalhos relacionados quanto à lógica *fuzzy* e os trabalhos com objetivos semelhantes em relação aos conjuntos por similaridade e busca por similaridade. O objetivo é apresentar o que já foi feito e em que pontos a pesquisa elaborada contribui para tais temas. Assim, na subseção 2.5.1 são apresentados os trabalhos relacionados em lógica *fuzzy* e na subseção 2.5.2 os trabalhos relacionados em similaridade.

2.5.1 Lógica *Fuzzy*

A lógica *fuzzy* é muito utilizada em diversas áreas dentro da computação e até em áreas de engenharia elétrica. Trabalhos como Chen (1995), Xu e Chen (2008) estudam métricas de distância para definir a similaridade entre conjuntos *fuzzy* intuicionistas e vagos (duas extensões da teoria de conjuntos *fuzzy*, quem podem ser reduzidas a esse), além disso, embora as métricas apresentadas pelos trabalhos visem medir a distância entre conjuntos, a teoria e equações utilizadas por eles podem medir a distância entre elementos de um mesmo conjunto.

Além disso, Silva *et al.* (2022) estuda formas de mineração de dados onde a localização é dada como um atributo *fuzzy*, de forma a possibilitar a elaboração de roteamento entre

tais pontos. Yager (2003) define métodos para criar sistemas de recomendação baseado em lógica *fuzzy*, baseado tanto nas preferências de usuário quanto no perfil e experiências passadas deste, enquanto Yera e Martinez (2017) realizam uma pesquisa com vários métodos de recomendação baseados em lógica *fuzzy*.

Todavia, nenhum dos trabalhos mencionados anteriormente combina a seleção de elementos com lógica *fuzzy* com a localização de facilidades, obtendo um resultado que visa auxiliar a resolução do PCV ao incluir a distância como variável *fuzzy* na elaboração do resultado.

2.5.2 Similaridade

Os *SimSets* foram elaborados por Pola *et al.* (2015) visando diminuir o número de itens em um banco de dados, e, para validar sua teoria, o autor utiliza os conjuntos por similaridade para encontrar quais as melhores estações meteorológicas e eliminar aquelas que fossem parecidas com essas, dessa forma, inicialmente os conjuntos por similaridade buscam encontrar os elementos semelhantes para eliminá-los da base de dados. Outros trabalhos, como Gonzaga e Cordeiro (2019) utilizam os conjuntos por similaridade para seleção de locais propícios para cultivo de plantas e seleção de animais para procriação com base no DNA, contudo, para fazer essa seleção é utilizado a divisão em conjuntos por similaridade (conceito similar à divisão nos conjuntos clássicos).

Ademais, Pola *et al.* (2015) define o algoritmo *Distinct*, que encontra o *SimSet* em um conjunto tradicional de dados complexos, utilizando para isso puramente a distância entre os elementos. Também, Alessi (2021) desenvolve o algoritmo *Asymmetric_Distinct*, uma variação do algoritmo anterior capaz de encontrar o *SymSet* onde as relações de similaridade não são simétricas, ou seja, o grafo de similaridade é direcionado.

Além disso, Tao *et al.* (2007) define um método para otimizar o desempenho do algoritmo RKNN em dados com muitas dimensões, visto que esse processamento é lento nesse tipo de dados. Gao *et al.* (2009) define uma nova variação do RKNN, onde há obstáculos entre objetos de forma que, mesmo que um objeto seja um dos k vizinhos mais próximos, ele não faça parte da resposta caso o caminho entre eles esteja bloqueado por um desses obstáculos, além de desenvolver um algoritmo para resolver tal problema.

Por fim, Santos *et al.* (2013) define uma forma de avaliar buscas por similaridade utilizando a diversidade dentro da resposta, visto que, várias vezes, os resultados podem ser muito parecidos, assim, a avaliação por diversidade pode suprir a falta de métricas externas para avaliar o resultado da busca.

Todavia, nenhum trabalho encontrado utiliza os *SimSet*s para selecionar elementos e obtê-los como resposta, em vez de obter o conjunto sem os dados semelhantes, sem utilizar divisão em conjuntos. Por fim, os métodos de extração de *SimSet*s são baseados puramente na distância, nenhum deles estuda a extração utilizando RKNN.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentadas as etapas utilizadas para a elaboração desta pesquisa. Inciando com os materiais utilizados, tanto em relação ao *software* quanto ao *hardware*, e em seguida, é definido o fluxo da pesquisa e desenvolvimento para cumprir os objetivos do trabalho definidos anteriormente na seção 1.1.

3.1 Materiais

Os materiais de *software* utilizados no desenvolvimento dessa pesquisa são:

- Linguagem de programação *Python*: foi utilizada para capturar, tratar e manipular os dados, além de descrever a lógica de programação; a versão utilizada é a 3.10.6.
- *Jupyter Notebooks*: a ferramenta foi utilizada junto da linguagem *Python* para tratamento de dados e visualização desses.
- Sistema gerenciador de banco de dados relacional (SGBDR) *PostgreSQL*: os dados tratados foram armazenados em um SGBDR para otimizar as consultas, minimizando o uso de memória principal e eliminando a necessidade de processamento dos dados a cada vez que o sistema seja iniciado.
- Sistema operacional (SO) Linux Mint 21.1: o SO responsável por gerenciar os recursos computacionais.

Para o desenvolvimento dessa pesquisa, foi utilizado um *Notebook* Dell G15 i1000-D20P, que tem como principais especificações de *hardware*:

- Processador Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz;
- 24 GB de memória RAM, divididas em dois módulos, um com 8 GB e outro com 16 GB.

Outras especificações podem ser obtidas consultando o modelo do *Notebook* de acordo com a fabricante.

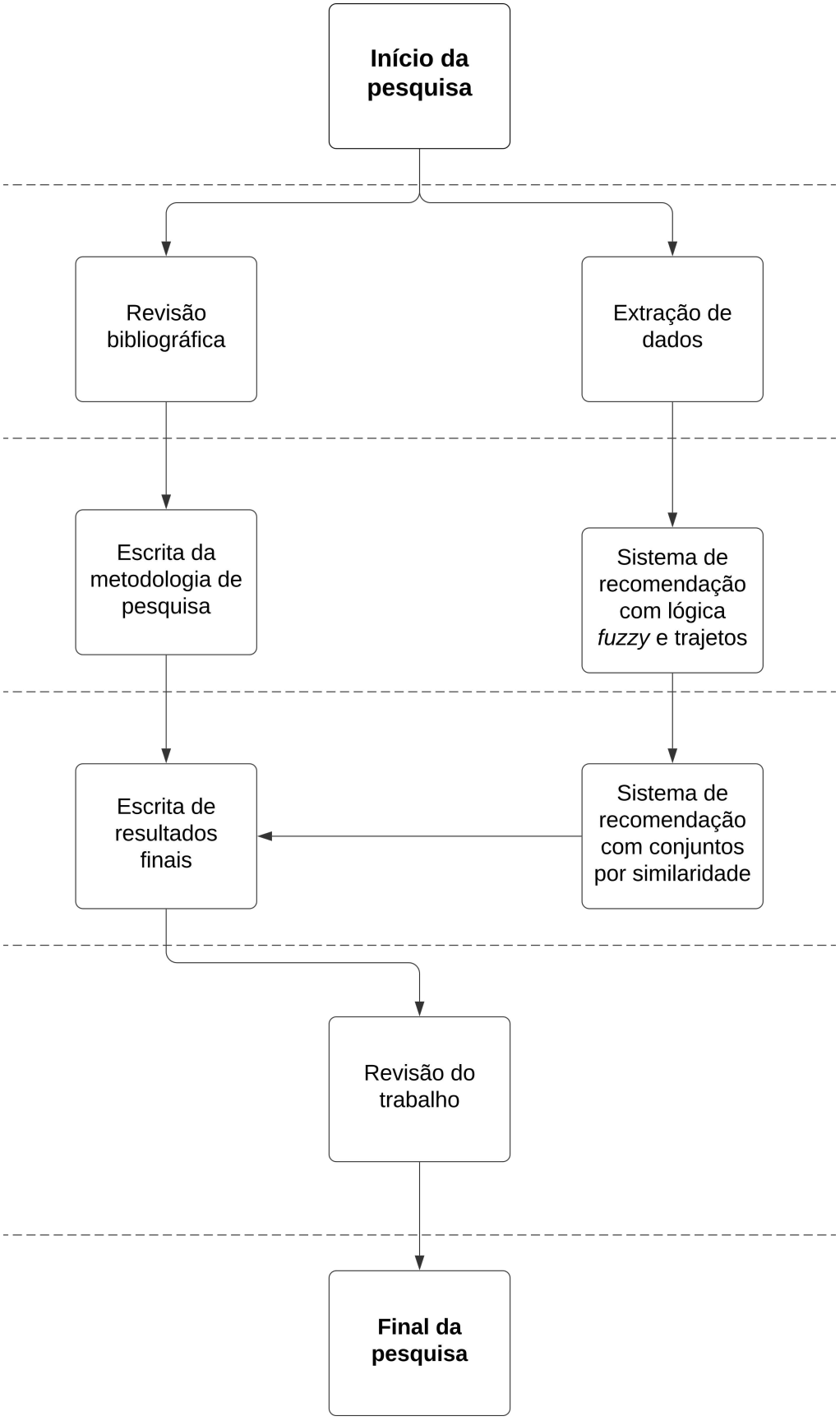
3.2 Métodos

A Figura 3 ilustra o fluxo de etapas para o desenvolvimento do trabalho, que serão discutidas a seguir.

As etapas de pesquisa teórica são:

- Revisão bibliográfica: esta etapa ocorreu durante todo o fluxo da pesquisa, envolvendo a investigação dos conceitos e trabalhos relacionados à abordagem teórica.

Figura 3 – Fluxograma de metodologia de pesquisa



Fonte: Autoria própria (2023).

- Escrita da metodologia: nesta etapa, é realizada a descrição de como a pesquisa foi desenvolvida, delineando as diferentes etapas de pesquisa e formas de avaliação dos resultados.
- Escrita de resultados finais: com a conclusão das etapas de metodologia e desenvolvimento, os resultados encontrados foram analisados e detalhados na monografia.
- Revisão do trabalho: após a conclusão de todas as etapas, o trabalho foi revisado, quanto à escrita, conceitos e implementação, com a correção de eventuais erros.

Além das etapas teóricas, o trabalho contêm etapas práticas, são elas:

- Extração de dados: os dados foram coletados e tratados, para eliminar inconsistências como localização errada ou valores faltantes. Essa etapa foi avaliada considerando a quantidade de itens encontrados para cada categoria de dados.
- Desenvolvimento de sistema de recomendação utilizando lógica *fuzzy* e trajetos: buscou-se encontrar, com base em restrições de um usuário um restaurante que atendesse tais condições, evitando gerar uma rota com distância excessiva. Para avaliar essa etapa, a distância da rota encontrada foi comparada com a de um algoritmo guloso.
- Desenvolvimento de sistema de recomendação com conjuntos por similaridade: foi criado um conjunto por similaridade para, com a entrada de um restaurante, sugerir outros restaurantes semelhantes. Para avaliar essa etapa, o tempo de busca no *SimSet* foi comparado com o tempo de busca utilizando RKNN.

A seguir, as etapas de desenvolvimento do sistema são apresentadas de maneira mais detalhada.

3.2.1 Recomendação por Lógica *Fuzzy* e Trajetos

Para obter a recomendação de restaurante, considerando a entrada do usuário com suas restrições e outros pontos de interesse, é necessário seguir algumas etapas, as quais são descritas a seguir.

Inicialmente, é necessário criar os conjuntos *fuzzy*, considerando a *fuzzyfication interface*. Cada restaurante deve ter graus de pertencimento em cada categoria de restaurante considerada. Apenas os restaurantes foram considerados como conjuntos *fuzzy*, já que as hospedagens e pontos turísticos são a entrada do usuário.

Em seguida, foi buscado encontrar os restaurantes que melhor atendam às condições do usuário, considerando a distância. Isso é feito utilizando os conceitos de lógica *fuzzy* apresentados na seção 2.2.

Por fim, os restaurantes encontrados devem ser inseridos na rota, buscando a menor distância. Para isso, será utilizada a métrica de distância geodésica, já que a localização dos pontos é dada por coordenadas de latitude e longitude. Esse processo será otimizado utilizando meta-heurísticas para aprimorar a qualidade da resposta.

3.2.2 Recomendação por Conjuntos de Similaridade

Para realizar a busca por similaridade, inicialmente, os restaurantes foram mapeados com suas características transformadas em um espaço métrico n-dimensional, permitindo a avaliação das relações de distância.

Em seguida, foi desenvolvido um algoritmo não determinístico para criar o conjunto por similaridade dos dados e indexar cada elemento ao outro que o representa. Durante as buscas, cada elemento pode servir como entrada, e os outros elementos indexados ao mesmo elemento mais significativo compõe a resposta.

Para fins de comparação, foi implementado o algoritmo de RKNN por força bruta, de forma que o tempo de execução das buscas entre os dois métodos apresentados pôde ser comparado.

4 RESULTADOS

Nessa seção, será apresentado o resultado da pesquisa, avaliado através de um sistema de seleção, roteamento e mineração de relacionamento entre restaurantes, hospedagens e pontos turísticos. Inicialmente, o sistema desenvolvido será apresentado, e em seguida, os resultados obtidos em cada etapa serão demonstrados e discutidos. Por fim, será discorrido acerca de possíveis trabalhos futuros.

4.1 Escopo do Sistema

O sistema desenvolvido durante a pesquisa consiste em três grandes partes principais: seleção de um restaurante com base em características desejadas por um usuário, utilizando lógica *fuzzy* e as operações definidas na teoria; o roteamento entre a hospedagem, pontos turísticos e os restaurantes selecionados, buscando reduzir o tamanho da rota. Por fim a análise das características dos restaurantes para encontrar relações que possibilitam sugerir outros restaurantes com base em outro restaurante, o qual o usuário teria gostado.

Durante a etapa de seleção, as relações *fuzzy* são exploradas com base nas informações dos tipos dos restaurantes entre si, assim como nas classificação de preços e qualidade. Entretanto, informações obtidas com base na relação entre duas ou mais características (por exemplo a relação de custo benefício) não foram exploradas.

Em seguida, com a seleção de restaurantes realizada, o roteamento é executado, buscando uma rota para cada restaurante escolhido. A rota parte da hospedagem, passando por todos os pontos turísticos, eventualmente chegando ao restaurante, e retorna ao ponto de origem. O usuário não tem controle sobre em que parte da rota o restaurante se encontra.

Por fim, as relações de preço, qualidade e tipo do restaurante são mineradas, gerando um conjunto por similaridade que pode responder à entrada de um restaurante com outros restaurantes que compartilham semelhanças. Esse conjunto por similaridade é gerado previamente e é imutável, pois o sistema não trabalha com a inserção e remoção de elementos, nem com redistribuição de relações de similaridade.

Resumidamente, o sistema desenvolvido simula uma aplicação que recebe como entrada as características de um restaurante desejadas (preço, qualidade e tipo), a localização onde o usuário está hospedado e os pontos turísticos que ele deseja visitar. Além disso, o sistema também pode receber como entrada um restaurante que o usuário gostou e responder com aqueles que a aplicação avalia como parecidos ao da entrada.

4.2 Resultados Obtidos

A seguir, os resultados para cada etapa serão apresentados. As etapas apresentadas são: coleta dos dados, mineração de relações *fuzzy* e seleção de elementos, busca e otimização da rota, construção e uso de conjuntos por similaridade.

4.2.1 Extração dos Dados

A coleta dos dados foi feita utilizando a *Application Programming Interface* (API) do Google Maps. No entanto, devido ao baixo número de respostas por requisição e ao limite de número de retornos, foi necessário buscar os estabelecimentos por com base nos bairros da cidade escolhida, no caso, NYC.

A escolha de NYC como cidade se deu pela grande quantidade de habitantes, fama global e por ser um ponto turístico amplamente conhecido, permitindo maximizar a quantidade de dados obtidos.

Para a busca de dados, foi utilizado o código apresentado no Apêndice A. O retorno das requisições foi tratado para remover os dados cujo a latitude fosse maior que 41° ou menor que 40.5°. O número total de dados obtidos após o tratamento pode ser visualizado na tabela Tabela 2.

Tabela 2 – Número de dados encontrados para cada tipo de estabelecimento

Tipo de estabelecimento	Número de dados
Hospedagem	655
Pontos Turísticos	582
Restaurantes	1467

Fonte: Autoria própria (2023).

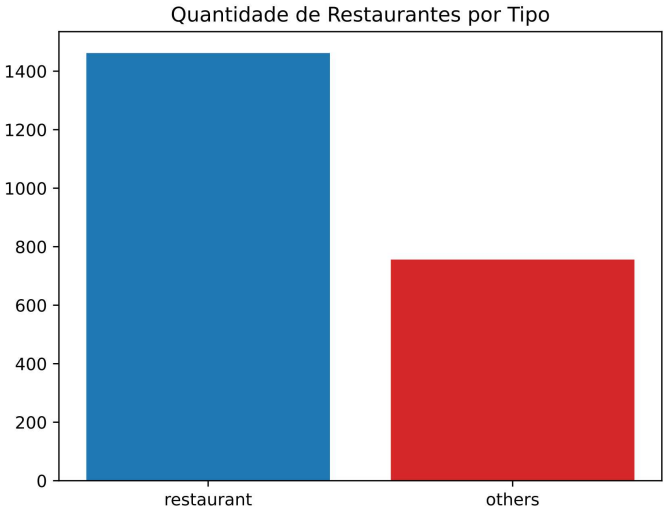
O grande número de dados, principalmente o de restaurantes, encontrados é suficiente para prosseguir com a pesquisa, já que esses dados foram utilizados em todas as etapas seguintes do desenvolvimento. Além disso, os restaurantes abrangem diversas categorias, incluindo uma classe geral denominada “Restaurante”. A comparação entre a quantidade de restaurantes com a classe “Restaurante” é apresentada na Figura 4

Observa-se na Figura 4 que a categoria “Restaurante” está presente em mais elementos do que as outras categorias juntas. Isso ocorre porque a classe “Restaurante” é mais abrangente e inclui vários elementos que possam não estar em mais nenhuma classe. A Figura 5 apresenta a comparação de quantidade de restaurantes nas outras categorias.

Dessa forma, é possível perceber que a categoria mais presente entre as outras é “Bar”, com mais de 350 estabelecimentos encontrados. Categorias como “Atração Turística” possuem poucos estabelecimentos que também são restaurantes.

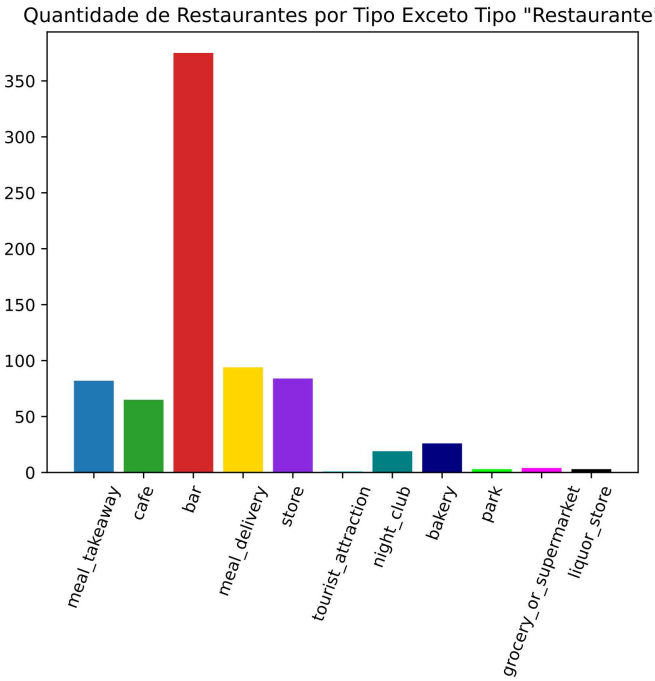
É importante destacar a diferença entre a categoria “Restaurantes” e os restaurantes considerados no trabalho. Enquanto a categoria abrange uma parte significativa de estabele-

Figura 4 – Comparação de quantidade entre categoria restaurante e outras categorias



Fonte: Autoria própria (2023).

Figura 5 – Comparação de quantidade entre categorias sem a categoria “Restaurante”



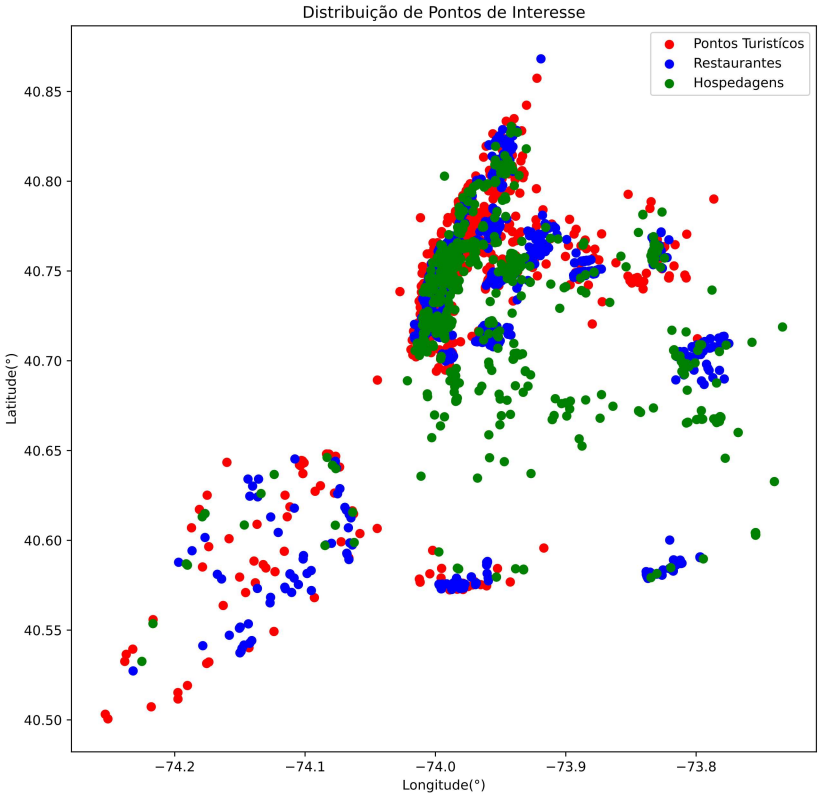
Fonte: Autoria própria (2023).

cimentos, os restaurantes, no propósito da aplicação, são o conjunto de estabelecimentos que trabalham com comida. A mesma lógica acontece com pontos turísticos e a categoria de restaurantes chamada “Atração turística”. Isso acontece pela forma como os dados requisitados são devolvidos e os nomes de categorias foram preservados (apenas sendo traduzidos, visto que a resposta das requisições é em inglês) para manter consistência.

Ademais, a localização dos estabelecimentos pode ser visualizada na Figura 6. Nessa figura, é possível perceber o formato da cidade, com pontos nas regiões onde se encontram

os bairros *Staten Island*, *Brooklyn*, *Queens* e *Manhattan*. O contorno do bairro *Bronx* não é perceptível na imagem.

Figura 6 – Localização das instâncias



Fonte: Autoria própria (2023).

4.2.2 Mineração de Relações *Fuzzy* e Seleção de Elementos

Após a extração e tratamento de dados, os pontos turísticos e hospedagens são representados apenas por suas coordenadas, enquanto o conjunto de restaurantes é representado como tuplas contendo as seguintes informações:

- Coordenadas;
- Qualidade (entre 1 e 5);
- Preço (entre 1 e 5);
- Tipo de restaurante (uma lista com todos os tipos que o restaurante satisfaz, podendo ser mais de um tipo).

A lista de tipos foi transformada em colunas de sim e não (booleanas), representando se o restaurante satisfaz ou não o tipo da coluna. Dessa forma, um restaurante fictício que possui

Tabela 3 – Exemplo de tipos de restaurante para restaurante fictício

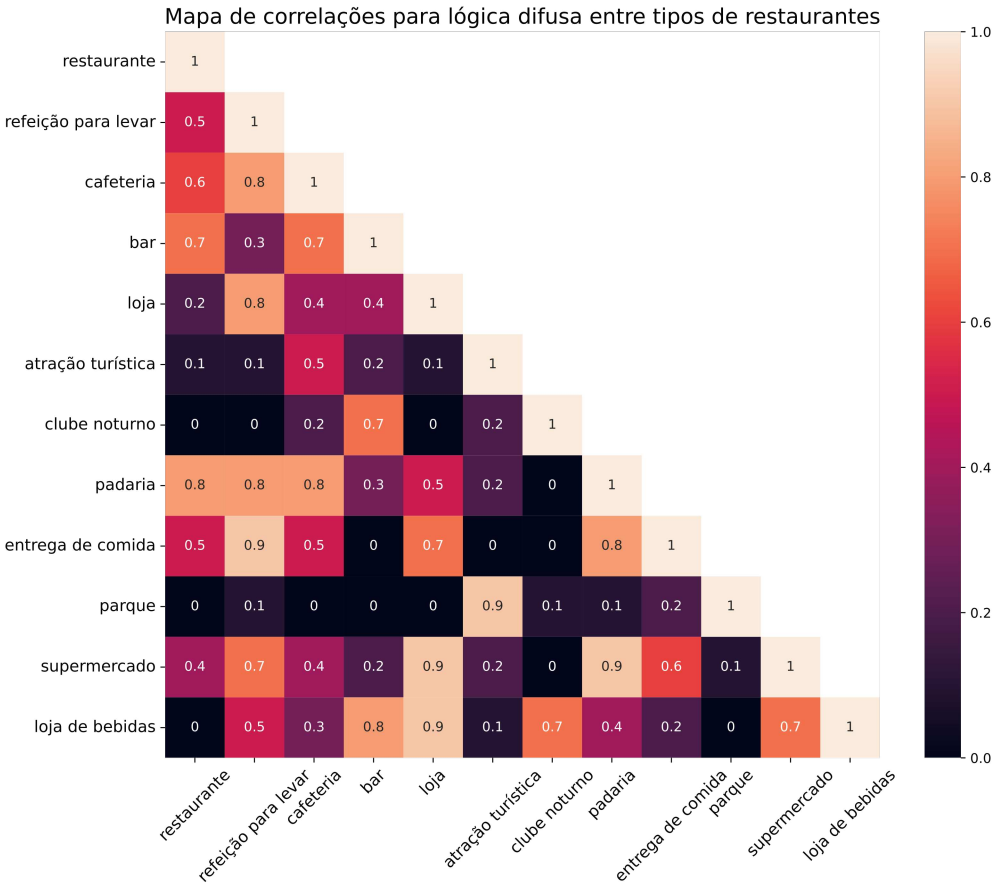
Cafeteria	Bar	Padaria	Entrega de comida	Loja
1	0	1	1	0

Fonte: Autoria própria (2023).

os tipos “cafeteria”, “padaria” e “entrega de comida”, mas não possui os tipos “bar” e “loja” é representado como na Tabela 3, onde 1 significa pertence e 0 significa não pertence

Após essa transformação, foi elaborada a *Fuzzyfication Interface* para os tipos de restaurantes. Os tipos deixam de valer 0 ou 1 e passam a ter graus de pertencimento, representados por valores no intervalo [0, 1]. Assim, um restaurante que pertence ao determinado tipo, possui grau 1, um restaurante que não pertence ao tipo possui grau 0, e os outros podem ter seus graus de pertencimento entre esses valores. Para mapear os dados, os tipos possuem relações entre si, conforme apresentado na Figura 7.

Figura 7 – Correlação entre tipos de restaurantes



Fonte: Autoria própria (2023).

O valor de cada tipo de restaurante é determinado pelo valor do tipo com maior relação (ou seja, valor mais alto) ao qual o restaurante que é mapeado pertence. Dessa forma, segue-se a relação de união definida por Zadeh (1965). O resultado do mapeamento é armazenado juntamente com os dados no SGBDR para possibilitar futuras consultas.

4.2.2.0.1 Seleção de Dados

Com os dados já mapeados, é possível realizar a seleção dos dados utilizando as regras definidas por Zadeh (1965) e Lee (1990), visando buscar os restaurantes que melhor satisfazem o critérios do usuário. O algoritmo para selecionar os dados pode ser visto no Apêndice B.

4.2.3 Busca e Otimização da Rota

A busca pela rota foi realizada utilizando o método GRASP, conforme apresentado no Apêndice C. O GRASP é uma meta-heurística que segue a lógica de um algoritmo guloso, porém introduzindo um fator probabilístico. Em vez de selecionar apenas o elemento mais próximo, esse método escolhe aleatoriamente um dos elementos mais próximos para incluir no percurso. Em seguida, a estratégia de busca local utilizada foi o algoritmo 2-opt, conforme apresentado no Apêndice D.

A rota obtida foi comparada em relação ao tempo de execução de um algoritmo de roteamento guloso, conforme apresentado no Apêndice E. Os resultados dessa comparação estão detalhados na Tabela 4, que apresenta uma análise comparativa entre os algoritmos, considerando tempo e qualidade de resposta. A descrição das instâncias pode ser vista na Tabela 5, na qual os índices representam o índice (ou posição) de cada elemento no banco de dados.

Tabela 4 – Comparação entre algoritmo GRASP e algoritmo guloso para a primeira instância

Instância	Melhor distância GRASP (Km)	Distância média GRASP (Km)	Tempo de execução médio GRASP (s)	Distância algoritmo guloso (Km)	Tempo de execução algoritmo guloso (s)
1	30.4014	30.4014 $\pm$ 7.1054e-15	0.0242 $\pm$ 0.0018	42.8616	1.4781e-5
2	53.9659	53.9752 $\pm$ 0.0651	0.0263 $\pm$ 0.0018	88.3557	1.5736e-5
3	54.8535	54.8602 $\pm$ 0.0293	0.0520 $\pm$ 0.0030	89.5085	1.9789e-5
4	82.466	82.5067 $\pm$ 0.1061	0.0226 $\pm$ 0.0017	127.6044	1.5259e-5
5	37.9569	37.9569 $\pm$ 1.42109e-14	0.0290 $\pm$ 0.0016	67.7030	1.5259e-5
6	53.4882	53.4882 $\pm$ 1.4211e-14	0.0268 $\pm$ 0.0028	82.7000	2.0504e-5
7	63.8291	63.8291 $\pm$ 2.8422e-14	0.0229 $\pm$ 0.0013	87.9774	1.4544e-5
8	43.2765	43.2765 $\pm$ 2.1316e-14	0.0267 $\pm$ 0.0017	66.4135	1.5497e-5
9	25.3066	25.3066 $\pm$ 3.5527e-15	0.0265 $\pm$ 0.0019	38.5772	1.5020e-5
10	43.9588	43.9588 $\pm$ 7.1054e-15	0.0276 $\pm$ 0.0014	74.2574	1.5497e-5

Fonte: Autoria própria (2023).

Com base nas Tabelas 4 e 5, observa-se que, embora o tempo de execução do algoritmo GRASP seja maior que o algoritmo guloso, ele apresenta uma resposta melhor em termos de qualidade.

Tabela 5 – Descrição das instâncias

Instância	Índice dos pontos turísticos	Índice da hospedagem	Coordenadas do restaurante
1	281, 132, 501, 103, 369, 427, 227, 148, 390, 126	158	40.7176 -73.9985
2	173, 19, 300, 157, 383, 382, 186, 345, 536, 44	424	40.7176 -73.9985
3	173, 19, 300, 157, 383, 382, 186, 345, 536, 44	424	40.7240 -74.0104
4	405, 377, 497, 511, 331, 299, 540, 565, 244, 431	232	40.7024 -73.9895
5	60, 258, 73, 520, 210, 424, 43, 384, 0, 216	219	40.7176 -73.9985
6	81, 565, 115, 412, 116, 119, 420, 465, 374, 384	480	40.7024 -73.9895
7	487, 88, 163, 231, 404, 105, 478, 533, 343, 19	589	40.7170 -73.9590
8	221, 519, 319, 389, 141, 1, 202, 551, 315, 259	212	40.7176 -73.9985
9	239, 289, 208, 62, 104, 437, 76, 461, 143, 293	485	40.7024 -73.9895
10	105, 46, 165, 182, 560, 371, 303, 150, 361, 264	469	40.7024 -73.9895

Fonte: Autoria própria (2023).

4.2.4 Construção e Uso de Conjuntos por Similaridade

Para recuperar um restaurante a partir de outros, foi criado um *SimSet*, no qual grupos de elementos são representados por outros elementos que são similares aos demais do grupo. Essa estratégia foi implementada para otimizar a velocidade de busca dos elementos.

Para a criação do *SimSet*, foi utilizado um algoritmo baseado em RKNN, apresentado no Apêndice F. A estratégia consiste em permitir que os elementos com maior frequência de presença nos vizinhos mais próximos possam referenciar os outros elementos dos quais fazem parte.

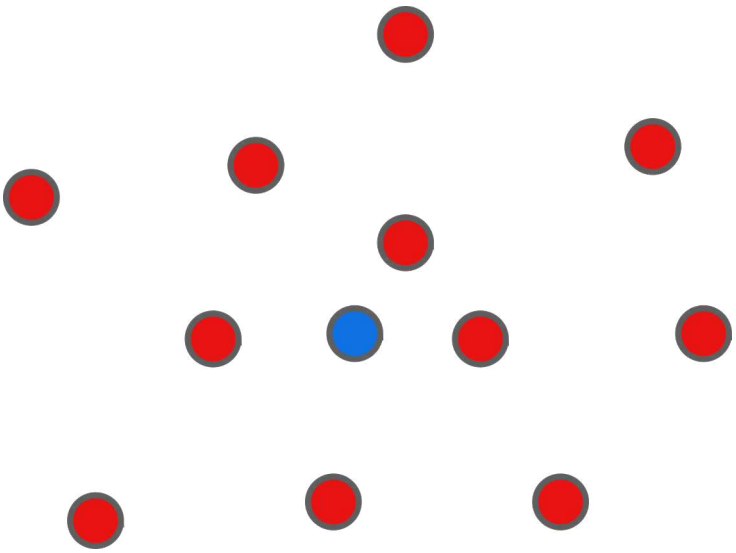
O algoritmo desenvolvido utiliza uma abordagem inovadora para gerar *SimSets*. O procedimento consiste em encontrar, em cada iteração, o elemento mais influente ainda no conjunto. Isso é feito selecionando o elemento que aparece mais vezes nos k vizinhos mais próximos dentro do conjunto. Ao selecionar o elemento mais influente, ele é atribuído como representante do conjunto e os elementos que o possuem como um de seus k vizinhos mais próximos são indexados a ele. Esse processo é repetido até que não exista outro elemento no conjunto. As Figuras 8 e 9 ilustram a construção do *SimSet* baseado em RKNN, supondo $k = 1$.

Na Figura 8, o elemento destacado em azul é o vizinho mais próximo de 4 elementos, assim, ele é o elemento mais influente.

Após identificar o elemento mais influente na Figura 8, a etapa apresentada na Figura 9 consiste em remover do conjunto os elementos destacados em azul e em cinza (que possuem o elemento azul como seu vizinho mais próximo) e inseri-los no conjunto por similaridade. Os elementos em cinza são referenciados no conjunto por similaridade pelo elemento em azul. Esse procedimento resulta em um *SimSet* gerado com base na influência dos elementos, diferentemente do *SimSet* gerado pelo método proposto em Pola *et al.* (2015), que considera apenas a distância de similaridade suficiente no conjunto.

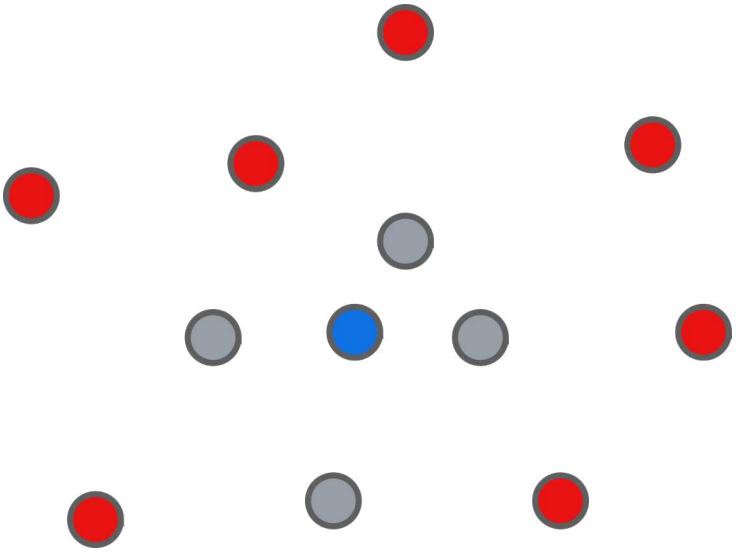
O tempo de execução da busca recuperando elementos utilizando o *SimSet* foi comparado com o tempo de recuperação do algoritmo RKNN, considerando as distâncias entre elementos calculada e armazenada previamente, visto que essa etapa é comum à construção

Figura 8 – Primeira etapa da construção do *SimSet*



Fonte: Autoria própria (2023).

Figura 9 – Segunda etapa da construção do *SimSet*



Fonte: Autoria própria (2023).

do *SimSet*. A comparação entre o tempo de execução entre os algoritmos pode ser visto na Tabela 6. O algoritmo para recuperação de dados por RKNN pode ser visto no Apêndice G.

Na Tabela 6, observa-se que o tempo de busca utilizando o *SimSet* é consideravelmente menor que o tempo de busca por RKNN. Isso ocorre principalmente porque os resultados já estão calculados, bastando encontrar qual elemento representa o conjunto em que o elemento de referência se encontra e devolver os outros itens desse mesmo conjunto.

Por fim, é importante ressaltar que o método utilizado resulta em uma aproximação do RKNN. Portanto, em tarefas que envolvem encontrar sobreposição ou quando o resultado exato é necessário, o método desenvolvido com *SimSets* não é adequado.

Tabela 6 – Comparação de tempo de execução entre *SimSet* e RKNN

Elemento buscado	**Tempo médio de execução com *SimSet* (s)**	**N° de elementos encontrados com *SimSet***	**Tempo médio de execução com RKNN (s)**	**N° de elementos** encontrados com RKNN
0	2.9385e-5 $\pm$ 9.591e-5	15	273.5643e-5 $\pm$ 41.5358e-5	14
1	3.0832e-5 $\pm$ 2.2605e-5	10	274.0288e-5 $\pm$ 16.7613e-5	0
2	2.4619e-5 $\pm$ 0.3344e-5	6	274.0674e-5 $\pm$ 13.2982e-5	0
3	2.0385e-5 $\pm$ 1.1875e-5	2	294.5127e-5 $\pm$ 24.4959e-5	0
4	1.2081e-5 $\pm$ 0.2037e-5	11	303.5614e-5 $\pm$ 35.622e-5	9
5	1.2641e-5 $\pm$ 0.4523e-5	10	317.1759e-5 $\pm$ 53.4292e-5	1
6	1.1172e-5 $\pm$ 0.2788e-5	16	331.8369e-5 $\pm$ 75.2259e-5	0
7	1.2925e-5 $\pm$ 0.3841e-5	19	298.3003e-5 $\pm$ 18.6406e-5	0
8	1.1737e-5 $\pm$ 0.267e-5	18	295.423e-5 $\pm$ 23.8219e-5	10
9	1.0743e-5 $\pm$ 0.1183e-5	17	283.1349e-5 $\pm$ 21.1769e-5	2

Fonte: Autoria própria (2023).

5 CONCLUSÃO

O trabalho proposto visava abordar dois desafios em sistemas de recomendação: a sugestão de itens baseado em lógica *fuzzy*, integrada à recomendação de uma rota otimizada, e a criação de conjuntos por similaridade para retornar elementos semelhantes a uma entrada específica. Na primeira etapa, foi desenvolvido um método utilizando combinação de interseções *fuzzy* e *fuzzyfication interface* para os dados dos elementos, identificando o elemento que melhor atende às exigências de um usuário. Na segunda etapa foi desenvolvido um método inovador não determinístico para extrair conjuntos de similaridade utilizando RKNN.

Com os resultados encontrados durante o desenvolvimento de pesquisa, observa-se que a busca por elementos que satisfaçam características específicas utilizando lógica *fuzzy* é viável, esse método demonstra eficácia, principalmente quando as características desejadas são muito restritivas, pois a estratégia utilizada permite encontrar os elementos que estão mais próximos de satisfazê-las. Além disso, a abordagem de resolução do PCV e problemas de roteamento semelhantes por meio do algoritmo GRASP se mostrou bastante eficaz, com resultados melhores que os de um algoritmo guloso; no entanto, a resolução com o algoritmo GRASP demanda mais tempo. Por fim, a recuperação de dados por similaridade utilizando *SimSet* é mais eficiente que a recuperação por RKNN, uma vez que seu tempo de execução é menor, além disso, o tempo de criação do *SimSet* não precisa ser considerado, pois, uma vez concluída, a construção se mantém, e a inserção de novos itens poderá ser gerenciada por outras estratégias, de forma incremental, como *leave* ou *replace* definidas na literatura.

Em todas as fases, os resultados obtidos satisfazem os objetivos almejados do trabalho. Foi possível selecionar o restaurante que melhor se adéqua às restrições de usuário, encontrar rotas mais curtas utilizando o método GRASP em relação ao algoritmo guloso e criar um conjunto por similaridade que retorna mapeamentos mais uniformes em relação ao número de elementos da resposta que o RKNN e em menos tempo. No entanto, o uso do GRASP foi provado ser muito mais lento que o algoritmo guloso, portanto, no futuro, outras meta-heurísticas como *Simulated Annealing* e colônia de formigas podem ser testadas, verificando a relação de tempo e qualidade de resposta. Além disso, o algoritmo de extração de *SimsSets* desenvolvido não considera se a escolha de um elemento pode excluir outro que seja igualmente importante e semelhante a outros elementos do conjunto.

Assim, o trabalho desenvolvido apresenta contribuições em sistemas de recomendação baseados em lógica *fuzzy*, PCV e conjuntos por similaridade, ao utilizar todos esses conceitos em um sistema prático e funcional.

REFERÊNCIAS

ALESSI, A. E. I. R. V. P. **Desenvolvimento de novas técnicas de extração de conjuntos de similaridade para relações não simétricas**. 2021. Disponível em: http://repositorio.utfpr.edu.br/jspui/handle/1/27551.

BLACK, P. E. **greedy algorithm**. Dictionary of Algorithms and Data Structures, 2005. Disponível em: https://xlinux.nist.gov/dads/HTML/greedyalgo.html. Acesso em: 11/10/2023.

BRITANNICA, T. E. o. E. 2023. Disponível em: https://www.britannica.com/science/NP-complete-problem. Acesso em: 09/10/2023.

CHEN, S.-M. Measures of similarity between vague sets. **Fuzzy Sets and Systems**, v. 74, n. 2, p. 217–223, 1995. ISSN 0165-0114. Disponível em: https://www.sciencedirect.com/science/article/pii/0165011494003399.

FILHO, E. **Cobertura por vértices mínima em grafos lei de Potência**. 2016. Disponível em: https://acervodigital.ufpr.br/handle/1884/45019.

GAO, Y. *et al.* Visible reverse k-nearest neighbor queries. *In*: **2009 IEEE 25th International Conference on Data Engineering**. [*S.l.*: *s.n.*], 2009. p. 1203–1206.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)**. First edition. [*S.l.*]: W. H. Freeman, 1979. ISBN 0716710455.

GAZDA, M. **TSP algorithms: 2-Opt, 3-opt in python**. 2019. Disponível em: http://matejgazda.com/tsp-algorithms-2-opt-3-opt-in-python/. Acesso em: 11/10/2023.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of Metaheuristics**. 2nd. ed. [*S.l.*]: Springer Publishing Company, Incorporated, 2010. ISBN 1441916636.

GONZAGA, A. dos S.; CORDEIRO, R. L. The similarity-aware relational division database operator with case studies in agriculture and genetics. **Information Systems**, v. 82, p. 71–87, 2019. ISSN 0306-4379. Disponível em: https://www.sciencedirect.com/science/article/pii/S0306437917303484.

GUTIN, G.; YEO, A.; ZVEROVICH, A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. **Discrete Applied Mathematics**, v. 117, n. 1, p. 81–86, 2002. ISSN 0166-218X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0166218X01001950.

JOHNSON, D. S.; MCGEOCH, L. A. The traveling salesman problem: A case study in local optimization. *In*: . [*s.n.*], 2008. Disponível em: https://api.semanticscholar.org/CorpusID:208903251.

KIM, B.-I.; SHIM, J.-I.; ZHANG, M. Comparison of tsp algorithms. **Project for Models in Facilities Planning and Materials Handling**, v. 1, p. 289–293, 1998.

LEE, C. Fuzzy logic in control systems: fuzzy logic controller. i. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 20, n. 2, p. 404–418, 1990.

LIPSCHUTZ, S.; LIPSON, M. **Matemática Discreta - 3ed: Coleção Schaum**. Bookman Editora, 2013. ISBN 9788565837781. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788565837781/. Acesso em: 30/10/2023.

LIU, Y. **Sistema de recomendação dos amigos na rede social online baseado em Máquinas de Vetores Suporte**. mar. 2014. 90 p. Dissertação (Mestrado) — Universidade de Brasíla, Brasília, mar. 2014. Disponível em: https://repositorio.unb.br/handle/10482/16533.

MENEZES, P. **Matemática Discreta para Computação e Informática - 4ed: Volume 16 da Série Livros didáticos informática UFRGS**. Artmed Editora, 2013. (Ufrgs). ISBN 9788582600252. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582600252/. Acesso em: 30/10/2023.

PAPADIAS, D.; TAO, Y. Reverse nearest neighbor query. *In*: ____. **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 2434–2438. ISBN 978-0-387-39940-9. Disponível em: https://doi.org/10.1007/978-0-387-39940-9_318. Acesso em: 16/10/2023.

POLA, I. *et al.* Similarity sets: A new concept of sets to seamlessly handle similarity in database management systems. **Information Systems**, v. 52, p. 130–148, 2015. ISSN 0306-4379. Special Issue on Selected Papers from SISAP 2013. Disponível em: https://www.sciencedirect.com/science/article/pii/S0306437915000216.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. *In*: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 283–319. ISBN 978-1-4419-1665-5. Disponível em: https://doi.org/10.1007/978-1-4419-1665-5_10.

RICCI, F. *et al.* **Recommender systems handbook**. New York; London: Springer, 2011.

ROCKI, K.; SUDA, R. Accelerating 2-opt and 3-opt local search using gpu in the travelling salesman problem. *In*: **2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)**. [*S.l.*: *s.n.*], 2012. p. 705–706.

SANTOS, L. F. *et al.* Evaluating the diversification of similarity query results. **Journal of Information and Data Management**, v. 4, n. 3, p. 188–188, 2013.

SILVA, H. P. da *et al.* Discovery ofnbsp;spatial association rules fromnbsp;fuzzy spatial data. *In*: **Conceptual Modeling: 41st International Conference, ER 2022, Hyderabad, India, October 17–20, 2022, Proceedings**. Berlin, Heidelberg: Springer-Verlag, 2022. p. 179–193. ISBN 978-3-031-17994-5. Disponível em: https://doi.org/10.1007/978-3-031-17995-2_13.

SINGH, A.; FERHATOSMANOGLU, H.; TOSUN, A. c. High dimensional reverse nearest neighbor queries. *In*: **Proceedings of the Twelfth International Conference on Information and Knowledge Management**. New York, NY, USA: Association for Computing Machinery, 2003. (CIKM '03), p. 91–98. ISBN 1581137230. Disponível em: https://doi.org/10.1145/956863.956882.

TAO, Y. *et al.* Multidimensional reverse knn search. **The VLDB Journal**, Springer-Verlag, Berlin, Heidelberg, v. 16, n. 3, p. 293–316, jul 2007. ISSN 1066-8888. Disponível em: https://doi.org/10.1007/s00778-005-0168-2. Acesso em: 16/10/2023.

WU, W. *et al.* Finch: Evaluating reverse k-nearest-neighbor queries on location data. **Proc. VLDB Endow.**, VLDB Endowment, v. 1, n. 1, p. 1056–1067, aug 2008. ISSN 2150-8097. Disponível em: https://doi.org/10.14778/1453856.1453970.

XU, Z.; CHEN, J. An overview of distance and similarity measures of intuitionistic fuzzy sets. **International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems**, v. 16, p. 529–555, 08 2008.

YAGER, R. R. Fuzzy logic methods in recommender systems. **Fuzzy Sets and Systems**, Elsevier, v. 136, n. 2, p. 133–149, 2003.

YERA, R.; MARTINEZ, L. Fuzzy tools in recommender systems: A survey. **International Journal of Computational Intelligence Systems**, Atlantis Press BV, v. 10, n. 1, p. 776, 2017.

ZADEH, L. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, 1965. ISSN 0019-9958. Disponível em: https://www.sciencedirect.com/science/article/pii/S001999586590241X.

ZEZULA, P. *et al.* Similarity search - the metric space approach. *In*: **Advances in Database Systems**. [*s.n.*], 2005. Disponível em: https://api.semanticscholar.org/CorpusID:41746867.

APÊNDICES

APÊNDICE A – Algoritmo para requisição de dados

Este apêndice apresenta o algortimo utilizado para recuperar os dados através da API do Google Maps.

Entrada: bairros B, cliente C, tipo T
Saída: arquivo de estabelecimentos E
Início:

- Para cada b em B faça:
  - para i no intervalo [0, 3[, faça:
    - consulta ← T + *in* + 'New York'
    - se i = 0, então:
      - lugares ← C.places(consulta)
    - senão:
      - lugares ← C.places(consulta, próxima página)
    - próxima_página ← lugares.get('next_page_token')
    - escreva lugares em E

Fim

Listagem 3 – Algoritmo para reuqisição de dados através da API do *Google Maps*

No algoritmo, o cliente é um objeto gerado para a API, e possui os métodos *places* e *get*; o método primeiro retorna os lugares encontrados e as suas informações, o segundo é utilizado para retornar o identificador da próxima página, de forma que, na próxima iteração do programa, os resultados sejam diferentes.

Além disso, o código executa apenas 3 iterações por bairro porque a API retorna apenas 60 lugares (20 por iteração).

APÊNDICE B – Código para mineração de relações *fuzzy*

Este apêndice apresenta o algoritmo para escolher elementos com base em lógica *fuzzy*, utilizado para obter os resultados parciais da subseção 4.2.2. O algoritmo é apresentado a seguir.

Entrada: pontos turísticos PT, hospedagem H, tipo T, preço máximo PR, qualidade mínima Q.
Saída: conjunto de restaurantes R.
Início:

1. Elimine os restaurantes onde o tipo *fuzzy* for maior que 0,5.
2. Elimine os restaurantes onde o preço for maior que PR.
3. Elimine os restaurantes onde a qualidade for menor que Q.
4. Converta o preço e qualidade para seus valores *fuzzy*, utilizando a Equação 4 e Equação 5.
5. Calcule e armazene a interseção dos valores *fuzzy* entre tipo, preço e qualidade, utilizando a Equação 3.
6. Elimine os dados onde a interseção é menor que 0,5 (esse passo é opcional e tem como propósito diminuir a densidade de cálculos para as próximas etapas)
7. Para cada restaurante, encontre a maior distância entre esse restaurante e algum dos pontos turísticos e armazene-a como distância.
8. Utilize a maior e menor distâncias máximas, encontradas anteriormente, de todo o conjunto para converter para uma variável *fuzzy*.
9. Elimine os dados onde a distância *fuzzy* for menor que 0,5.
10. Calcule a interseção (utilizando novamente a Equação 3) entre a distância *fuzzy* e a interseção anterior.
11. Retorne os dados cuja nova interseção calculada for igual à maior do conjunto.

Fim

Listagem 4 – Algoritmo de Seleção de Restaurantes com lógica *fuzzy*

A conversão dos valores para o valor *fuzzy* é dada com a fórmula:

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{4}$$

$$f = mx + (y_1 - m * x_1) \tag{5}$$

Para cada conversão, as equações acima foram utilizadas considerando os valores das variáveis como:

- Preço:

  - x_1: menor preço do conjunto;
  - x_2: preço máximo do requisito de usuário;
  - y_1: 1;
  - y_2: 0,5;
  - x: preço a ser convertido.

- Qualidade

  - x_1: menor qualidade do requisito;
  - x_2: maior qualidade no conjunto de dados;
  - y_1: 0,5;
  - y_2: 1;
  - x: qualidade a ser convertida.

- Distância:

  - x_1: menor distância;
  - x_2: maior distância;
  - y_1: 1;
  - y_2: 0,5;
  - x: distância a ser convertida.

Em todos os cálculos, o valor convertido é representado por f.

APÊNDICE C – Algoritmo GRASP

Entrada: grafo G, alfa a
Saída: distancia D, rota R
Início:

- atual ← tamanho do grafo - 1
- primeiro ← atual
- rota ← lista(atual)
- distância ← 0
- enquanto tamanho do grafo ≠ tamanho da rota, faça:
  - linha ← grafo[atual]
    nós_disponíveis ← lista()
  - para cada i em tamanho da linha, faça:
    - se i não está na rota e linha[nó] < infinito, então:
      - adicione o nó ao fim de nós disponíveis
  - maior_distância ← 0
  - para cada i em tamanho de nós_disponíveis, faça:
    1. se linha[nós_disponíveis[i]] > maior_distância, então:
       a) maior_distância ← linha[nós_disponíveis[i]]
  - gama ← alfa * maior_distância
  - valores_restritos ← lista()
  - para cada nó em nós_disponíveis, faça:
    - se nó < gama, então:
      1. inclua o nó no fim de valores_restritos
  - se tamanho de valores_restritos > 0, faça:
    - viagem ← escolha_aleatória(valores_restritos)
  - senão:
    - viagem ← escolha_aleatória(nós_disponíveis)
  - distância ← distância + grafo[atual][viagem]
  - adicione viagem ao fim da rota
  - atual ← viagem
- distância ← distância + grafo[atual][primeiro]
- adicione o primeiro ao fim da rota
- retorne a distância e a rota

Fim

Listagem 5 – Etapa construtiva do método GRASP para PCV

O algoritmo a representa a etapa construtiva do método GRASP utilizado para resolver o PCV. Esse procedimento deve ser seguido de uma etapa de otimização, como o algoritmo 2-opt.

No algoritmo, o parâmetro alfa corresponde ao fator probabilístico, de forma que, ao aproximar esse valor de 0, o algoritmo se comporta como um algoritmo completamente aleatório, enquanto ao aproximar o valor de 1, o algoritmo se comporta como o algoritmo guloso. Em todos os testes, esse valor foi fixado como 0,7.

APÊNDICE D – Algoritmo 2-opt

Esse apêndice apresenta o algoritmo 2-opt, utilizado para otimizar a rota gerada como resposta da etapa construtiva do método GRASP, mostrada no Apêndice C.

Entrada: grafo G, rota R
Saída: distância D, rota melhorada M
Início:

- r ← cópia de R
- melhor_rota ← r
- melhor_distância ← calcula_distância(G, r)
- melhorado ← verdadeiro
- enquanto melhorado = verdadeiro, faça:
  - melhorado ← falso
  - para i em [1, tamanho de r -2[, faça:
    - para j em [i + 1, tamanho de r[, faça:
      - nova_rota ← *swap*(grafo, nova_rota)
      - nova_distância ← calcula_distancia(G, nova_rota)
      - se nova_distância < melhor_distância, então: atualize a rota e melhorado
  - r ← melhor_rota
- retorne melhor_distancia, melhor_rota

Fim

Listagem 6 – Algoritmo 2-opt

Esse algoritmo visa melhorar uma rota já criada, fazendo as trocas entre arestas da rota (o que é feito dentro da função *swap*). Esse é um algoritmo custoso, já que sua complexidade é $O(n^2)$.

APÊNDICE E – Algoritmo Guloso

Esse apêndice apresenta o pseudo-código do algoritmo guloso utilizado para resolver o problema do caixeiro viajante.

Entrada: grafo G
Saída: distância D, rota R
Início:

- atual ← tamanho de G - 1
- rota ← lista(atual)
- distância ← 0
- enquanto o tamanho da rota < tamanho do grafo, faça:
  - linha ← grafo[atual]
  - menor_distância ← infinito
  - índice_menor_distância ← infinito
  - para cada i em [0, tamanho da linha[:
    - se linha[i] < menor_distância e i não está na rota:
      - menor_distância ← linha[i]
      - índice_menor_distância ← i
    - adicione o índice_menor_distância no final da rota
    - atual ← índice_menor_distância
    - distância ← menor_distância
  - distância ← distância + grafo[atual][tamanho do grafo -1]
  - adicione o item em tamanho do grafo -1 no fim da rota

  retorne a rota

Fim.

Listagem 7 – Algoritmo guloso para resolver o PCV

Esse algoritmo procura sempre o nó mais próximo disponível para viajar, assim, o resultado não considera passos futuros, apenas o melhor no momento. Embora seja muito rápido, a qualidade da resposta é geralmente pior que a resposta de outras meta-heurísticas como GRASP.

APÊNDICE F – Construção do *SimSet*

Este apêndice apresenta o algoritmo para encontrar um *SimSet* baseado em RKNN. O pseudo-código é apresentado a seguir:

Entrada: matriz de similaridade M
Saída: *SimSet* S
Início:

- S ← lista()
- pode_ser_avaliado ← lista()
- para cada i em [0, tamanho de M[, faça:
  - adicione verdadeiro à pode_ser_avaliado
- enquanto houver verdadeiro em pode_ser_avaliado, faça:
  - mais_influente, frequência ← encontra_mais_influente(M, pode_ser_avaliado)
  - se houver mais_influente, então:
    - pare
  - conectados ← encontre_conectados(M, mais_influente, pode_ser_avaliado)
  - se pode_ser_avaliado[mais_influente], então:
    - adicione [mais_influente, mais_influente] a S
    - pode_ser_avaliado[mais_influente] ← falso
  - para cada i em [0, tamanho de pode_ser_avaliado[:
    - se i em conectados, então:
      - adicione [i, mais_influente] a S
      - pode_ser_avaliado[i] ← falso
- para cada i em [0, tamanho de pode_ser_avaliado[, faça:
  - se pode_ser_avaliado[i], então:
    - adicione [i,i] a S
- retorne S

Fim

Listagem 8 – Algoritmo para a construção de *SymSets* baseado em rklnn

No algoritmo acima, a matriz de similaridade indica para cada elemento do conjunto quais são os k elementos mais próximos, assim, se calculada previamente, economiza processamento. Além disso, a função encontra_mais_influente retorna qual elemento ainda não avaliado aparece mais vezes na matriz de similaridade e sua quantas vezes ele aparece; a função

encontre_conectados encontra todos os elementos ainda não avaliados que estão conectados a um elemento.

Por fim, a última iteração garante que itens pouco influentes sejam inseridos ao *SimSet*, indexados a si mesmos.

APÊNDICE G – Algoritmo *Reverse K Nearest Neighbours*

Esse apêndice contém o pseudo-código para executar o algoritmo RKNN por método de força bruta, o algoritmo é apresentado a seguir.

Entrada: matriz de similaridade M, item I
Saída: elementos similares E
Início:

- similares $\leftarrow$ lista()
- para cada linha em M:
  - se item em lista, então:
    - adicione número da linha em similares
- retorne similares

Fim

Listagem 9 – Algoritmo de busca por RKNN

Esse algoritmo consiste em verificar todos os elementos em uma matriz que contém os k vizinhos mais próximos e retornar todos os elementos que contém o elemento de busca como seus *K Nearest Neighbors* (KNN). O uso de uma matriz de similaridade visa economizar tempo na execução, visto que é muito custoso fazer esse cálculo a cada iteração.