UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ALCEU CICHACZEWSKI

PROPOSTA DE ROTA PARA A COLETA DO RESÍDUO URBANO ORGÂNICO DE UM BAIRRO DA CIDADE DE FRANCISCO BELTRÃO - PR

FRANCISCO BELTRÃO

2019

ALCEU CICHACZEWSKI

PROPOSTA DE ROTA PARA A COLETA DO RESÍDUO URBANO ORGÂNICO DE UM BAIRRO DA CIDADE DE FRANCISCO BELTRÃO - PR

Trabalho de Conclusão de Curso de Especialização, apresentado ao Curso de Especialização em Métodos Matemáticos Aplicados, da Universidade Tecnológica Federal do Paraná, Campus Francisco Beltrão, como requisito parcial para a obtenção do Título de especialista em Métodos Matemáticos Aplicados.

Orientador: Prof. Dr. Vilmar Steffen.
Coorientador: Prof. Me Franklin Angelo Krukoski.

FRANCISCO BELTRÃO

2019

TERMO DE APROVAÇÃO

Trabalho de Conclusão de Curso de Especialização

PROPOSTA DE ROTA PARA A COLETA DO RESÍDUO URBANO ORGÂNICO DE UM BAIRRO DA CIDADE DE FRANCISCO BELTRÃO - PR

por

ALCEU CICHACZEWSKI

Trabalho de Conclusão de Curso de Especialização apresentado às 13 horas e 30 min. do dia 09 de novembro de 2019, como requisito parcial para obtenção do grau de especialista em Métodos Matemáticos Aplicados, da Universidade Tecnológica Federal do Paraná, Campus Francisco Beltrão. O candidato foi arguido pela Banca Avaliadora composta pelos professores que abaixo assinam este Termo. Após deliberação, a Banca Avaliadora considerou o trabalho _____ (Aprovado ou Reprovado).

VILMAR STEFFEN
Professor Orientador

FRANKLIN ÂNGELO KRUKOSKI
Professor Coorientador

MAIQUIEL SCHMIDT DE OLIVEIRA
Membro da Banca

Prof. Vilmar Steffen
Responsável pela Coordenação do CEMMA
Curso de Especialização em Métodos Matemáticos Aplicados

A FOLHA DE APROVAÇÃO ORIGINAL (ASSINADA) ENCONTRA-SE NA COORDENAÇÃO DO CURSO DE ESPECIALIZAÇÃO EM MÉTODOS MATEMÁTICOS APLICADOS.

“Dedico este trabalho à minha família, e aos meu colegas de curso pelos momentos de ausência, e pelo auxilio na sua conclusão”.

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Vilmar Steffen e coorientador Prof. Me. Franklin Angelo Krukoski, por me aturarem na conclusão desse trabalho.

Aos meus colegas do curso de Engenharia de Computação da UTFPR – PB, por me ajudarem a concluir esse trabalho.

Gostaria de deixar registrado também, o meu reconhecimento e agradecimento especial ao professor Dalcimar Casanova, por contribuir com o algoritmo que tornou possível a realização desse trabalho.

“*O sucesso não consiste em não errar, mas em não cometer os mesmos equívocos mais de uma vez*.” (George Bernard Shaw)

RESUMO

Observa-se que com o crescimento cada vez maior das cidades, acabam por surgir também, diversos problemas desde ordem social a de saúde pública. Dentre os diversos problemas de saúde pública, podemos destacar um em especial, a coleta dos resíduos sólidos orgânicos, produzidos por essa população. Tal serviço é de extrema necessidade, pois quando não atende a toda uma demanda, acaba por causar sérios problemas de saúde pública, como proliferação de doenças e animais indesejáveis. Desta forma, neste trabalho desenvolveu-se um estudo para propor uma rota, para a realização de forma eficiente e eficaz do serviço de coleta de resíduo orgânico, e procurando para tanto atender a todos os domicílios, de um bairro da cidade de Francisco Beltrão, com o intuito de prever se possível a redução de custos, devido á redução do deslocamento realizado pelo caminhão de coleta. A partir dos dados obtidos, como mapa do bairro e tamanho das ruas do mesmo, fornecidos pela secretaria de meio ambiente do município de Francisco Beltrão-PR, obteve-se a distância entre cada esquina (vértice) do referido mapa (grafo) em estudo, e com a implementação do algoritmo do carteiro chinês, em conjunto com o algoritmo de Dijkstra implementados em Python, propôs-se uma rota, que visa melhorar a eficiência da coleta dos resíduos, no referido bairro de estudo que atenda todos os domicílios. A partir deste estudo realizado não foi possível garantir que a rota obtida é a rota ótima. Sendo necessário maiores pesquisas e dados para obter o trajeto ótimo viável.

Palavras-chave: Otimização de rotas. Problema do carteiro chinês. Resíduos sólidos. Algoritmo Dijkstra. Python.

ABSTRACT

It is observed that with the increasing growth of cities, many problems arise from social order to public health. Among the various public health problems, we can highlight one in particular, the collection of organic solid waste produced by this population. Such a service is of extreme necessity, because when it does not meet all the demand, it ends up causing serious public health problems, such as proliferation of diseases and undesirable animals. Thus, in this work, a study was developed to propose a route for the efficient and effective implementation of the organic waste collection service, and seeking to serve all households in a neighborhood of the city of Francisco Beltrão. in order to predict if possible the cost reduction, due to the reduction of the displacement made by the collection truck. From the data obtained, such as neighborhood map and street size, provided by the environment secretary of the municipality of Francisco Beltrão-PR, the distance between each corner (vertex) of the map (graph) under study was obtained. , and with the implementation of the Chinese Postman algorithm, together with the Dijkstra algorithm implemented in Python, a route was proposed to improve the efficiency of waste collection in the study district that serves all households. From this study it was not possible to guarantee that the obtained route is the optimal route. Further research and data are required to obtain the optimal viable route.

Keywords: Route optimization. Chinese postman problem. Solid waste. Dijkstra algorithm. Python

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Observa-se que, com o passar dos anos a urbanização alcançou um grande crescimento populacional, algo visto não somente nos grandes centros, mas também nas pequenas cidades. Aliado a esse crescimento surgiram também vários problemas de saúde pública, um deles é como tratar a quantidade de resíduos produzidos por todo esse excedente populacional, surgindo assim a necessidade de se melhorar um serviço de coleta desse rejeito, que consiga atender a toda essa demanda populacional.

Sendo assim, o serviço de coleta de resíduos urbano orgânico consiste basicamente em um caminhão, adaptado para tal finalidade, que circula pelas ruas, nas quais ficam localizados os domicílios a serem atendidos e, com a ajuda de pessoas treinadas recolhe os resíduos orgânicos produzidos por cada família, e leva para o local onde este resíduo será depositado. Um grande problema observado com isso é que, esse veículo não possui uma rota pré-determinada, apenas é necessário que sejam percorridas todas as vias em que a coleta é feita, sendo que a rota fica a critério do motorista, muitas vezes realizando percursos desnecessários, o que acabam tornando a coleta prolongada e exaustiva, além de gerar custos que podem ser reduzidos a partir de um bom planejamento do trajeto.

Tendo em vista o que foi apresentado, este trabalho procurou analisar a rota de coleta urbana do resíduo orgânico, de um bairro do município de Francisco Beltrão localizado no estado do Paraná, procurando-se assim propor um percurso, em que o caminhão da coleta realize, tendo como objetivo redução (e se possível a minimização) de custos com relação a deslocamentos do veículo. Com isto, neste trabalho tem-se por objetivo fazer uso de ferramentas de programação linear implementada em Python, além da associação à teoria de grafos, para assim propor uma rota que os caminhões da coleta urbana devem fazer, para atender a todos os domicílios de um determinado bairro, percorrendo a menor distância possível, e reduzindo os gastos com recursos humanos e materiais, ocasionado assim uma melhor eficiência e qualidade no serviço prestado a toda a população do município.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Propor uma rota, para a coleta do resíduo urbano orgânico, no bairro da Cango no município de Francisco Beltrão – PR.

1.1.2 Objetivos específicos

- Fazer uma análise do mapa do bairro da Cango, no município de Francisco Beltrão, e com base neste sugerir uma rota para a coleta do resíduo urbano orgânico;
- Propor uma rota de coleta do resíduo, que percorra a menor distância possível, atendendo a todos os domicílios;
- Representar a rota, obtida por minimização da distância percorrida, e apresentar a mesma através de um grafo não orientado, onde ocorra a fácil visualização e aplicação a realidade do município.

2 REFERENCIAL TEÓRICO

2.1 DEFINIÇÃO DE RESIDUO SÓLIDO ORGÂNICO

O resíduo sólido orgânico, popularmente conhecido como lixo é um material utilizado pelo ser humano em diversas atividades, de industriais a domésticas, que após o seu uso já não possui mais valor para ser conservado. Além de ser considerado um rejeito inesgotável, esse material pode ser parcialmente utilizado, gerando entre outros aspectos, proteção à saúde pública e a economia de recursos naturais (NETO et al., 2007). De acordo com:

> A NBR–10004, assim define lixo: “resíduos nos estados sólidos e semissólidos, que resultam de atividades da comunidade de origem: industrial, doméstica, hospitalar, comercial agrícola, de serviços e de varrição” (NBR,2004).

Segundo essa mesma Norma NBR-10004, esses subprodutos são considerados inúteis ou indesejáveis, e cuja composição ou quantidade de líquido gerado, não os permita que escoe livremente pela natureza. Estes resíduos podem ser classificados como:(NETO et al., 2007)

a) Resíduos sólidos agrícolas - resíduos sólidos resultantes da criação e abate de animais e do processamento da produção das plantações e cultivos;

b) Resíduos sólidos industriais - resultantes dos processos industriais e das manufaturas;

c) Resíduos sólidos institucionais - originados dos serviços de saúde, educação, pesquisa e outros;

d) Resíduos sólidos municipais - resíduos residenciais e comerciais gerados pela comunidade do município;

e) Resíduos sólidos de pesticidas - os resíduos da manufatura, do manuseio e do uso de substâncias químicas para matar pestes, animais e vegetais;

f) Resíduos sólidos residenciais - resíduos que normalmente se originam no interior das residências, algumas vezes chamados resíduos sólidos domésticos;

Sabe-se ainda, que diariamente são coletadas no Brasil uma grande quantidade de resíduos sólidos urbanos, além disso essa produção de resíduos está em franca ascensão, apesar das grandes diferenças regionais, a produção de

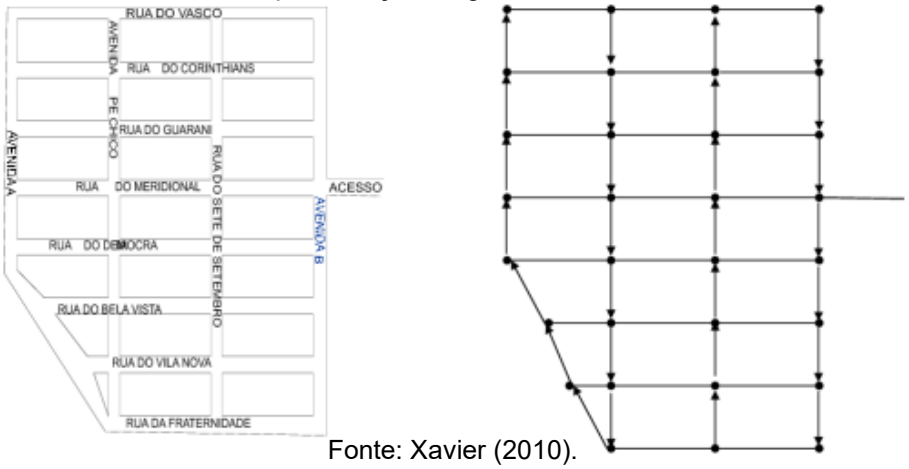
resíduos tem crescido em todas as regiões e estados brasileiros. Consequentemente, boa parte desses resíduos não possui destinação sanitária e ambiental adequada. Embora tenha havido progresso nos últimos anos, os resíduos ainda são depositados em vazadouros a céu aberto, os chamados lixões, em mais da metade dos municípios brasileiros (GOUVEIA, 2012).

Observando essa problemática no município de Francisco Beltrão localizado no sudoeste do Paraná, não seria diferente, o mesmo contava com uma população de 78943 habitantes no senso do IBGE de 2010, e previsão estimada para 2018 de 89942 habitantes (CIDADES, 2019). Com isso pressupõem-se que, a produção de resíduo urbano orgânico aumente com o passar dos anos. Sendo assim para se atender a toda essa demanda residencial, o município conta com uma frota de 9 caminhões, adaptados para realizar a coleta desse material, com colaboradores qualificados, que realizam tal atividade de forma a evitar a proliferação de doenças, insetos e animais indesejáveis e a contaminação do meio ambiente.

2.2 GRAFOS

Um grafo pode ser definido como uma representação gráfica, de interdependência entre elementos que são identificados como nós ou vértices. Esses elementos são simbolicamente interligados, através de um traço denominado aresta ou arcos, ocorre em alguns casos desses grafos possuírem ou não orientação, quando possuem tal propriedade eles são conhecidos como grafos orientados ou direcionados, e caso não possuam tal atributo são conhecidos como dígrafos. Além disso um grafo pode possuir um arco, que nada mais é que um vértice, onde uma aresta chega ao mesmo e outra aresta sai deste vértice. Um exemplo de grafo pode ser visto na Figura 1. As rotas de coleta de lixo são circuitos sobre o grafo, no caso orientado, começando e terminando no nó depósito/aterro (XAVIER, 2010).

Figura 1 - Exemplo de representação de um grafo, a esquerda mapa do bairro em estudo, a direita representação do grafo orientado.



Fonte: Xavier (2010).

Com relação aos grafos, podemos ainda verificar algumas propriedades presentes na grande maioria deles, por exemplo os nós em um grafo são chamados de adjacentes se forem os extremos de uma mesma aresta. Já um laço pode ser definido como uma aresta com extremos no mesmo nó. E duas arestas que contém os mesmos extremos são chamadas de arestas paralelas (XAVIER, 2010).

Sabendo-se ainda que um vértice pode ser interligado por N arestas, podemos definir como o grau de um vértice, o número de arestas incidentes a ele, por exemplo se existirem 2 arestas ligando 3 vértices, esses vértices possuem grau par (ROSEN, 2010). Além do conceito de grau de um grafo outro conceito importante é o caminho em um grafo, que nada mais é, que uma sequência de vértices começando em V_0 até V_k, e passando por arestas A_0 até A_k. Sendo assim podemos determinar que o comprimento de um caminho é o número de arestas que ele contém, e consequentemente se uma aresta for usada mais de uma vez, ela deve ser contada tantas vezes quantas ela for usada (XAVIER, 2010).

2.2.1 Método de Euler

O teorema de Euler, consiste basicamente em mostrar que um grafo conectado, que possui um caminho entre todos os pares de vértices, tem um circuito que passa exatamente uma vez por todas as suas arestas, se, e somente se, todos os vértices deste grafo tiverem, grau par. Os grafos nos quais existem roteiros que passam exatamente uma única vez por todas as arestas são denominados grafos

Eulerianos. Complementarmente, os roteiros formados dentro destes grafos são denominados: circuito ou ciclo Euleriano (FILHO; JUNQUEIRA, 2006).

2.2.1.1 Grafo Euleriano e Circuito Euleriano

Podemos definir um grafo euleriano ou um circuito euleriano, como um grafo não direcionado, que contém todas as arestas do grafo sem que ocorra a repetição da mesma aresta mais de uma vez (KONOWALENKO, 2012).

> TEOREMA – Um grafo contém um circuito euleriano, se, e somente se, o grafo não tem nenhum nó de grau ímpar(KONOWALENKO,2012).
>
> **Prova:**
> Seja G um grafo euleriano. Logo ele contém um circuito euleriano. Em cada ocorrência de vértice desse caminho existe uma aresta que chega nesse vértice e outra que sai desse vértice. Como toda aresta faz parte do caminho, isto é, nenhuma aresta fica fora do caminho, necessariamente o número de arestas incidentes em cada vértice é par. Como todo vértice possui grau par, então na construção de um caminho sempre é possível chegar e sair de um vértice passando por arestas diferentes, que ainda não tenham sido utilizadas (KONOWALENKO,2012).

2.2.3 Problema do Carteiro Chinês (PCC)

Pode ser definido como um problema que tem como objetivo cobrir com um percurso todos os arcos(vértices) do grafo, de maneira a minimizar a distância total percorrida, esse caminho se diferencia do circuito euleriano, por nele ser permitida, se necessário, a repetição de arestas, além disso, esse problema admite uma solução em tempo polinomial (MORO, 2014).

As principais abordagens do Problema do Carteiro Chinês são:

1. **Problema do Carteiro Chinês Não Direcionado (PCCND)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo $G = (V, A)$, a partir de um vértice $V_0 \in V$. Exemplo: Cidades somente com ruas de mão dupla (MORO, 2014).
2. **Problema do Carteiro Chinês Direcionado (PCCD)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo $G = (V, A)$, a partir de um vértice $V_0 \in V$. Exemplo: Cidades somente com ruas de mão única (MORO, 2014).
3. **Problema do Carteiro Chinês Misto (PCCM)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo $G = (V, A)$, a partir de um

vértice $V_0 \in V$. Exemplo: Cidades com ruas de mão dupla e mão única (MORO, 2014).

Basicamente, os diversos problemas do Carteiro Chinês se resumem em transformar o grafo original em um grafo euleriano para cada uma das versões em estudo (MORO, 2014).

2.2.3.1 Função objetivo do PCC

Como existem vários modelos matemáticos para cada problema do carteiro chinês, este trabalho procurou se restringir ao PCCND, que é o foco deste estudo, e seu modelo matemático é apresentado a seguir: (MORO et al., 2018).

Minimizar:

$$\sum_{i=1}\sum_{j=1} C_{ij}X_{ij} \qquad (1)$$

Sujeito a:

$$\sum_{k=1} X_{ki} - \sum_{k=1} X_{ik} = 0 \qquad \text{i=1,2,.....n} \qquad (2)$$

$$X_{ij} + X_{ji} \geq 1 \quad \forall \quad (i,j) \in A \qquad (3)$$

$$X_{ij} \geq 0 \text{ inteiras} \qquad (4)$$

Neste modelo matemático, a função objetivo, representada pela Equação (1), representa o custo total a ser minimizado, ou seja, no caso do trabalho em estudo, a distância total a ser percorrida, onde o termo X_{ij} (representa o número de vezes em que a aresta (*i*, *j*) é percorrido de *i* para *j*, e o termo C_{ij} (representa o comprimento ou custo da aresta (i,j).Com relação as restrições na Equação (2), esta garante a continuidade da rota, as restrições na Equação (3) garantem que nenhuma aresta deixará de ser considerada e, em relação a Equação(4), tem-se que as variáveis do problema são não negativas e inteiras (MORO et al., 2018).

Com relação a esse problema, pode-se observar ainda que o mesmo se modifica quanto a sua complexidade, dependendo do número de variáveis e restrições consideradas em sua formulação. Alguns podem ser classificados como intratáveis conforme seu tamanho. Mesmo utilizando computadores de última

geração, dificuldades podem ser encontradas, pois estas residem na natureza combinatória desse tipo de problema. Quando ocorrem tais situações complexas, pode-se utilizar técnicas para alcançar soluções próximas da ótima, como podemos citar os métodos de otimização heurística (MIURA, 2003).

Sendo assim o algoritmo que descreve o problema do carteiro chinês pode ser visto no Quadro 1:

Quadro 1 – Algoritmo do carteiro chinês não orientado.

```
Inicio
    Ler o Grafo G= (V, A);
    Se todos os nós em G, do grafo original, possuem grau par
        Então determinar um ciclo euleriano em G e Fim.
    Organizar um grafo Kn da seguinte forma:
        Reunir todos os vértices de grau ímpar no grafo Kn (grafo somente
        com os vértices de grau ímpar) e associar a cada par de vértices i e j
        no grafo, uma aresta (i,j) com peso igual ao caminho mais curto que liga
        i a j no grafo G.
    Determinar o menor percurso em Kn, e representar o mesmo por M*.
        Para cada aresta pertencente a M* associar uma nova
        aresta em G no caminho mínimo que ela representa,
        obtendo um grafo Ga.
    Determinar a solução do carteiro chinês que é representada
        por um ciclo euleriano em Ga.
FIM
```

Fonte: Goldbarg e Luna (2005).

2.3. MÉTODOS DE OTIMIZAÇÃO

Com relação aos métodos de otimização, eles podem ser determinísticos ou heurísticos, e as variáveis de decisão, podem ser discretas ou contínuas. As distribuições discretas são situações em que o espaço amostral contém um número finito ou infinito de pontos, porém contáveis, neste caso, uma variável X é denominada de variável aleatória discreta. Já quando o espaço amostral possui um número infinito de pontos, o mesmo será representado por distribuições contínuas

de probabilidade, no caso das variáveis continuas (SILVA, 2015). Sendo assim os problemas de otimização podem ser classificados em 2 grupos:

a) **Programação linear:** São problemas nos quais se busca a melhor alocação de recursos, de forma a atingir determinado objetivo de otimização, atendendo a determinadas restrições. Para tanto, pode-se utilizar como representação gráfica uma linha reta (CORRAR, 2004).
b) **Programação Não-Linear:** Já a PNL têm por finalidade, resolver problemas que envolvem funções constituídas de variáveis, que compartilham relações desproporcionais entre si (não-linearidade). Assim, utilizam-se os mesmos conceitos (otimização, função-objetivo, variáveis de decisão e restrições), embora os procedimentos matemáticos empregados na solução dos problemas de natureza não linear sejam diferentes (CORRAR, 2004).

2.3.1 Métodos Determinísticos

Nesta classe de métodos de otimização está a classe de métodos determinísticos, que consiste em utilizar valores da função objetivo e/ou do vetor gradiente e da matriz hessiana da referida função, como uma forma de guiar o processo de busca da região de soluções possíveis ao problema de otimização, além disso este método possui uma certa dificuldade em encontrar uma solução viável e não há garantia que a mesma seja ótima (CARDOSO; AVILA, 2013).

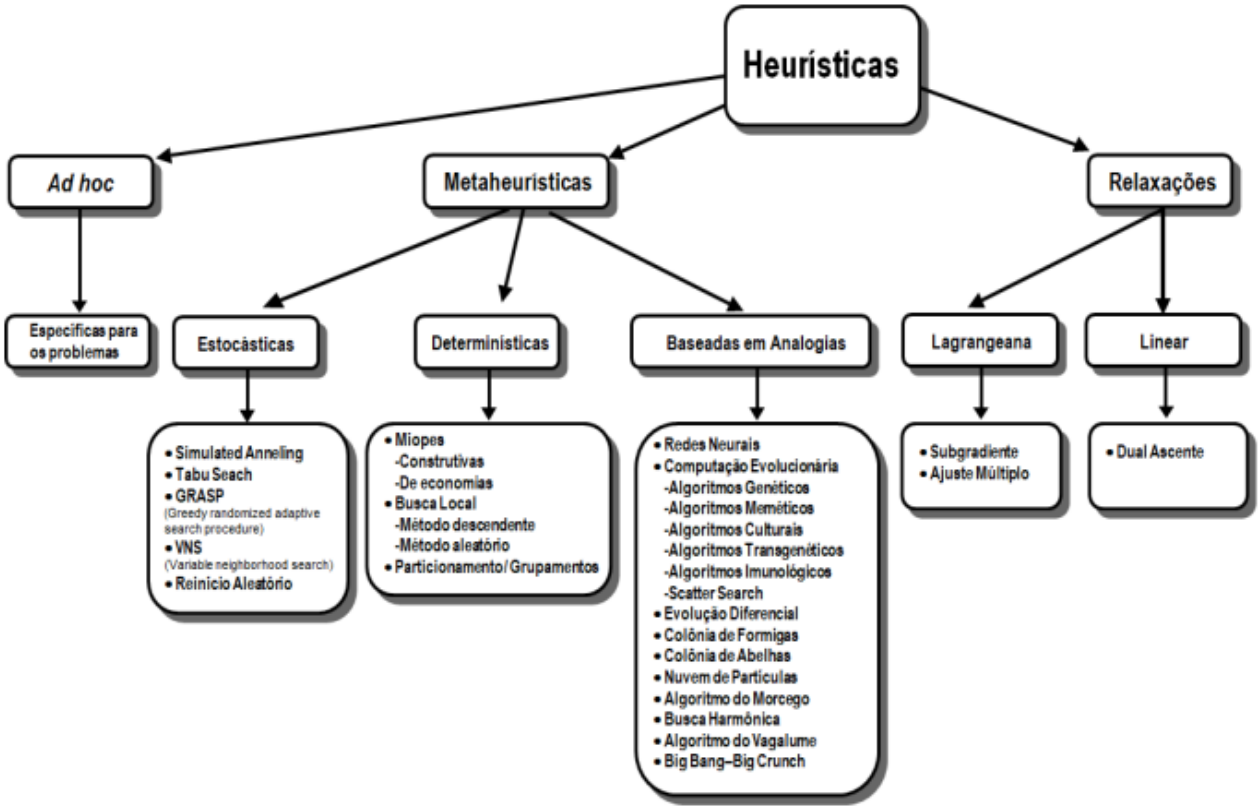
2.3.2 Método Heurístico

Observando-se que os métodos exatos existentes, não conseguem mais resolver determinados tipos de problemas, extremamente difíceis, surgiram os métodos heurísticos, que possuem uma maior chance de encontrar a solução ótima e grande facilidade de encontrar uma solução viável, para determinados problemas (JUNIOR, 2019).

O método heurístico, é definido como uma técnica que consiste, na obtenção de soluções para se delimitar qual a direção a ser seguida, que seja capaz de chegar a determinado objetivo, representando uma combinação de duas condições: a necessidade de simplificar os critérios, e de forma simultânea, a capacidade de se discernir dentre alternativas boas e ruins. Sendo assim é um método computacional

aproximado, que busca uma solução considerada aceitável para um dado problema, que pode ser representado computacionalmente, sendo esperado em muitos casos, que elas alcancem valores ótimos (JUNIOR, 2019). O método heurístico pode ser representado a partir da Figura 2:

Figura 2 - Classificação dos métodos heurísticos.



Fonte: Goldbarg et al.(2016).

2.3.3 Métodos Meta-Heurísticos

O termo metaheuristica é definido, como uma classe de algoritmos genéricos de busca, que podem ser utilizados em diferentes tipos de problemas. Este método se utiliza de ideias baseadas em analogias, para realizar o processo de busca da solução para problemas de otimização. Por exemplo, a metaheurística *Têmpera Simulada (*Simulated Annealing) é baseada em um processo físico da metalurgia. Outros exemplos são os *Algoritmos Genéticos* que são fundamentados em conceitos da evolução populacional (FREITAS et al., 2010).

O processo geral de execução de uma metaheurística, trata da busca de soluções a partir da visitação de regiões do espaço de soluções, guiando-se por

funções de satisfação. A função de satisfação é uma função matemática, que atribui um valor a cada solução do espaço de busca. Para comparação, considere uma técnica exaustiva: para determinar a melhor solução de um problema, todas as soluções existentes são visitadas e ao final a melhor é retornada (FREITAS et al., 2010).

Uma das principais aplicações da meta-heurística, são em problemas que se tem poucas informações para auxiliar no processo de resolução, em que a utilização de métodos exaustivos e repetitivos não é viável, e há poucos dados heurísticos para prosseguir. No entanto, caso uma provável solução seja encontrada, ela pode ser analisada para verificar sua viabilidade (JUNIOR, 2019). Entre os diversos métodos meta-heurísticos existentes, podemos destacar alguns:Enxame de partículas (Particle Swarm Optimization), Busca Harmônica (Harmonic Search), Algoritmo de Dijkstra que pertence á classe de algoritmos de busca local.

2.3.3.1 Enxame de partículas (Particle Swarm Optimization).

Inspirado no comportamento e na dinâmica dos movimentos dos pássaros ou peixes. O método consiste, em um sistema que é inicializado com uma população de soluções aleatórias com velocidades iniciais aleatória e procura por um resultado ótimo que envolve a reprodução de comportamento cognitivo e social dos grupos anteriormente citados. No PSO as soluções potenciais, chamadas de partículas, voam através do espaço do problema seguindo as então melhores partículas (que possuem os melhores valores no momento). O PSO se baseia na informação da trajetória das partículas (indivíduos), e dos pontos do espaço de busca visitados por elas, para informar a qualidade da solução (qualidade da função objetivo). Para tanto, usa-se uma estrutura de memória para preservar os melhores locais visitados. A indicação da movimentação de cada partícula, a cada nova iteração, depende de duas informações: a melhor posição de todo o enxame e a melhor posição da própria partícula (MEDEIROS, 2012).

2.3.3.2 Busca harmônica (Harmonic Search)

Consiste na observação do desempenho de músicos, em uma orquestra que buscam a harmonia perfeita. Na música, esta harmonia perfeita é considerada análoga, a achar a solução ótima de um problema de otimização, e refere-se a um

dado padrão de qualidade de áudio. O HS inspira-se na observação da capacidade de improvisação dos músicos, para a obtenção de novas harmonias, levando-se em conta a frequência, o timbre e a amplitude do instrumento de cada um deles. O método pode ser resumido em cinco passos (MEDEIROS, 2012):

a) Inicializar os parâmetros do problema e do algoritmo;

b) Inicializar a memória da harmonia;

c) Improvisar novas harmonias;

d) Atualizar a memória da harmonia e;

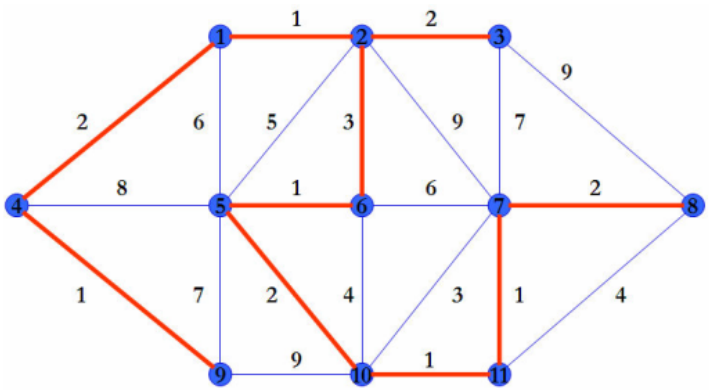
e) Verificar critério de parada.

2.3.3.3 Algoritmo de Dijkstra

O algoritmo de Dijkstra, tem como objetivo obter o menor caminho, entre um dado vértice fixo e todos os demais vértices do grafo (como por exemplo, saber a distância mínima entre uma loja e todos os seus clientes). Consiste basicamente em fazer uma visita por todos os nós do grafo, iniciando no nó fixo dado e encontrando sucessivamente o nó mais próximo, o segundo mais próximo, o terceiro mais próximo e assim por diante, um por vez, até que todos os nós do grafo tenham sido visitados (SANTOS,2010).

Na Figura 3 é apresentado um grafo como forma de ilustração para compreender o funcionamento do algoritmo de Dijkstra, partindo-se do ponto $V_i(9)$ e procurando as arestas de menor peso ou valor em todo o grafo, o algoritmo de Dijkstra procura assim chegar ao $V_f(8)$, para tanto o mesmo, necessita passar pelos vértices 9,4,1,2,3,2,6,5,10,11,7,8, que possuem as arestas com os menores pesos, obtendo assim a menor rota para o grafo apresentado.

Figura 3 – Grafo completo



Fonte: Barros, Pamboukian e Zamboni (2007).

Algoritmo:

Seja S o nó origem:

- Atribua valor zero à distância do nó S a ele mesmo (origem) e infinito às demais;

- Atribua um valor qualquer aos precedentes (o precedente de um nó T é o nó que precede T no caminho mínimo de S para T);

- Enquanto houver nó aberto;

- Seja K um nó ainda aberto cuja distância seja a menor dentre todos os nós abertos;

- Feche o nó K;

- Para todo nó J ainda aberto que seja sucessor de K faça:

- Some a estimativa do nó K com a distância do arco que une K a J;

- Caso esta soma seja menor que a distância anterior para o nó J, substitua-a e anote K como precedente de J.

Quando todos os nós tiverem sido fechados, os valores obtidos serão as distâncias mínimas que partem do nó origem até os demais nós da rede. O caminho propriamente dito é obtido a partir dos nós chamados acima de precedentes (SILVA; SANCHES, 2004).

No Quadro 2, podemos observar a representação do algoritmo de Dijkstra em pseudocódigo:

Quadro 2 - Algoritmo de Dijkstra

Procedimento *Dijkstra* (G:grafo simples conexo com peso, com todos os pesos positivos)

{G tem vértices $a=V_0,V_1,.....,V_n=Z$ *e pesos* $w(V_i,V_j)$

em que $w(V_i,V_j)=\infty$ se (V_i,V_j) não for uma aresta em G}

Para i:= 1 **até** *n*

$L(V_i) := \infty$

L(a) := 0

S := Ø

{os rótulos agora são inicializados de modo que o rótulo de *a* seja 0 e todos os outros sejam ∞, e S seja o conjunto vazio}

Enquanto $Z \notin S$

Início

u := um vértice não em S,com L(u) mínimo

$S := S \cup \{u\}$

Para todos os vértices v não em S

Se L(u) + w(u,v) < L(v) **Então** L(v) := L(u) + w(u,v) {isso adiciona um vértice a S com rótulo mínimo e atualiza os rótulos dos vértices que não estão em S}

Fim {L(z) = comprimento de um caminho mais curto de *a* a *z*}

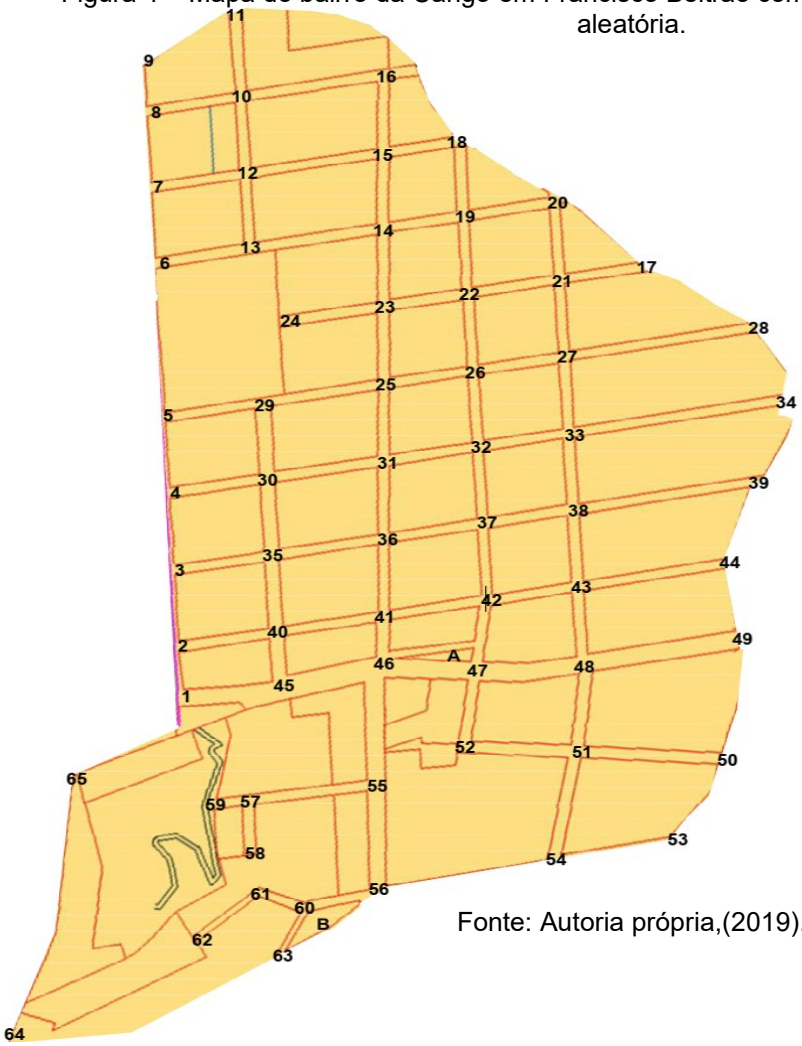
Fonte: Rosen (2010).

3 MATERIAIS E MÉTODOS

3.1 MATERIAIS

Neste trabalho, foi feita uma análise na rota de coleta de resíduos sólidos orgânico do bairro da Cango, um bairro da cidade de Francisco Beltrão-PR. O mapa do bairro foi fornecido pela secretaria de meio ambiente de Francisco Beltrão, secretaria responsável diretamente pela administração da coleta dos resíduos sólidos orgânicos. O serviço de coleta tem a sua disposição uma frota de 8 veículos, dentre os quais um é o veículo reserva. A coleta do material é feita em diferentes dias da semana em cada bairro, o horário da coleta também pode variar para os dias da semana em que são realizados. A coleta do resíduo no referido bairro de estudo ocorre nas segundas-feiras a partir das (12h40), nas quartas-feiras a partir das (04h30) e aos sábados a partir das (04h30) onde o veículo utilizado possui uma capacidade de carga de 12 Toneladas.

Figura 4 – Mapa do bairro da Cango em Francisco Beltrão com a indicação das arestas de forma aleatória.



Fonte: Autoria própria,(2019).

Os dados de distância entre cada vértice (pontos numerados na Figura 4), utilizados para propor a rota de coleta, encontram-se no Apêndice A. Os dados de latitude e a longitude de cada vértice, obtidos utilizando o Google Maps (medidas aproximadas), estão disponíveis no Apêndice B. A utilização do Google Maps foi necessária pois a secretaria de meio ambiente não possui estas informação de georreferenciamento.

3.2 MÉTODOS

Com base no mapa apresentado na Figura 4, será realizado um estudo para propor uma rota de coleta do resíduo orgânico urbano, sendo que este estudo tem como primeiro passo a numeração dos vértices, o que já foi feito, conforme pode ser visto na Figura 4 de posse desta primeira informação, ainda é necessário determinar as distâncias entre cada vértice, e as latitudes e longitudes obtidas para tais nós. Após a numeração dos vértices, fez-se a análise de quais desses vértices possuem grau ímpar e quais possuem grau par, sendo uma das condições necessárias para ocorrer a criação do ciclo euleriano, que será utilizado para realizar o desenvolvimento da rota. Então, utilizou-se um método de programação linear proposto por Edmonds e Johnson (1973) para resolver o problema do carteiro chinês, formulando o circuito euleriano. Entretanto, para que o método possa ser aplicado, é preciso que todos os vértices do grafo sejam de grau par, caso contrário é necessária a adição de alguns vértices no mesmo, de forma a tornar os mesmos de grau par. Para adição deste novos vértices, utiliza-se o método proposto por Edmonds e Johnson (1973) em conjunto com o algoritmo de Dijkstra, para determinar o conjunto de novos vérticos com menor custo (distância) total. A implementação desta rotina, em Python, encontram-se no Anexo A.

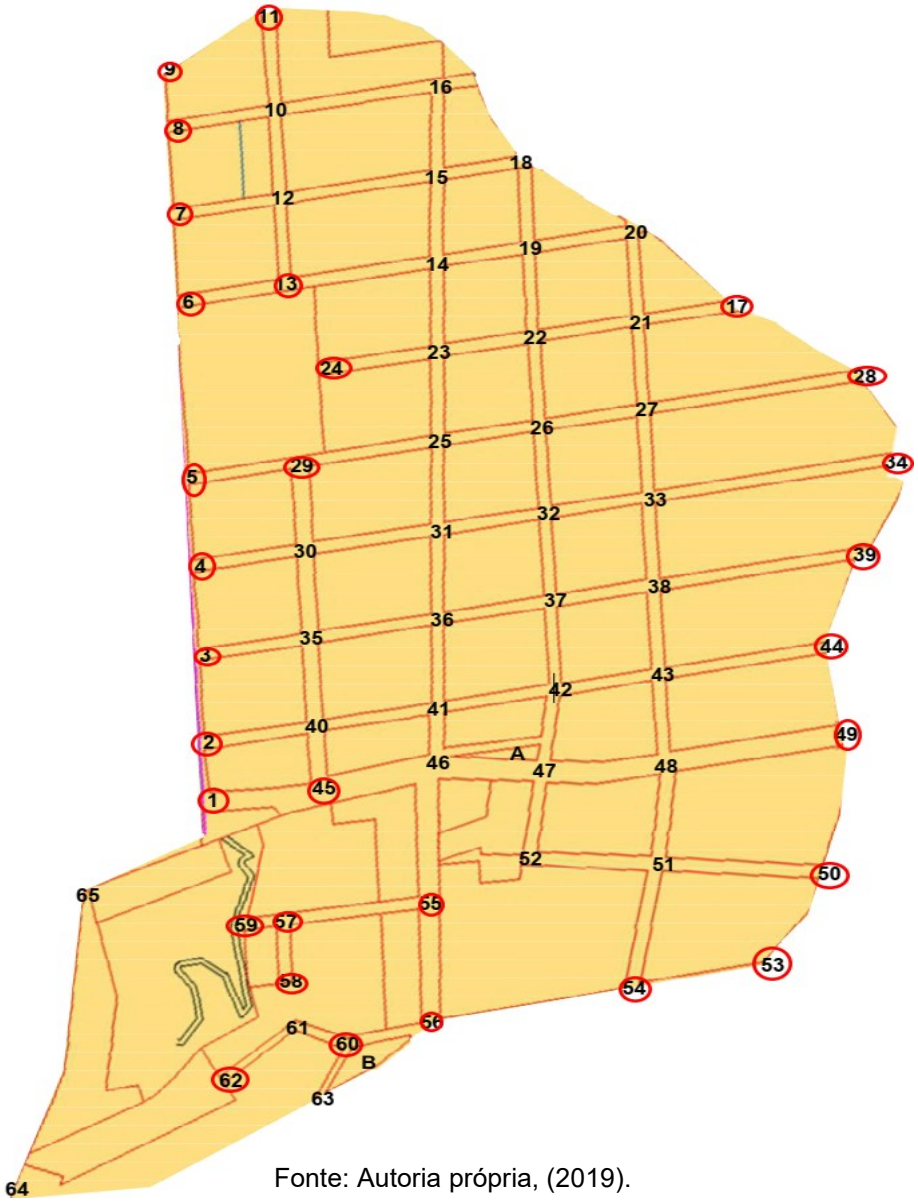
Para realizar a simulação foi utilizado o Google Colab (um ambiente virtual que não requer configuração, e é executado na nuvem e está disponibilizado de forma gratuita pela Google). Para ocorrer a correta realização desta simulação, é necessário enviar ao algoritmo os arquivos que contém as distâncias entre cada vértice e as latitudes e longitudes. As informações destes arquivos são apresentadas, respectivamente nos Apêndices A e B.Tais arquivos estão disponíveis no GitHub. A vantagem da utilização dessa ferramenta é que a mesma não possui qualquer limitação no número máximo de variáveis utilizadas para sua execução.

Com os resultados obtidos gera-se um grafo, onde é possível visualizar a rota proposta.

4 RESULTADOS E DISCUSSÕES

Para podermos propor a rota de coleta, foi necessário primeiramente verificar se o grafo fornecido com os vértices numerados aleatoriamente, possui todos os nós de grau par, condição necessária para aplicação do método proposto no trabalho de Edmonds e Johnson (1973). Entretanto, como pode ser observado na Figura 5, isso não ocorre, pois o grafo apresentado, de um total de 65 vértices, possui 30 vértices de grau ímpar que estão circulados em vermelho.

Figura 5 – Mapa original com os vértices de grau ímpar indicados em vermelho.



Fonte: Autoria própria, (2019).

Sendo assim, para fazer com que todos os vértices do grafo tenham grau par e, por consequência, o grafo seja euleriano, é necessário primeiramente fazer a

adição de alguns vértices que conectam os nós de grau ímpar. Como pode ser observado da Figura 5, no grafo em estudo existem 30 vértices de grau ímpar em todo o grafo e, como cada aresta conecta dois vértices, é necessária a adição de 15 arestas. Levando em consideração que, a ideia é ter uma rota com o menor percurso possível, por conseguinte a soma dos percursos das 15 arestas escolhidas deve ter o menor peso(percurso)possível, conectando todas os vértices de grau ímpar. Fazendo uma estimativa por análise combinatória, de quantas possibilidades devem ser analisadas, encontrou-se o número de 435 possibilidades

$$C= \frac{30!}{2!*(30-2)!} =435$$

Consequentemente foi utilizado ainda o algoritmo de Dijkstra para auxiliar na escolha das 15 arestas que tem o menor percurso total. Sendo que este percurso não faz parte do grafo original, ou seja, é um percurso adicionado apenas para que o método de Edmonds e Johnson (1973) possa ser aplicado. Esta “adição” de arestas é também uma adição no percurso total a ser realizado na rota, pois no circuito euleriano uma aresta é percorrida apenas uma vez, e estas arestas adicionadas representam o caminho que será percorrido mais de uma vez no circuito não euleriano.

As arestas conectando os parede de vértices são apresentadas no Quadro 3 e ilustradas na Figura 6, sendo que com a adição dessas arestas, o grafo original sofre um aumento no número total de arestas de 93 para 108:

Quadro 3 - Pares de vértices ímpares com a soma de arestas de menor peso.

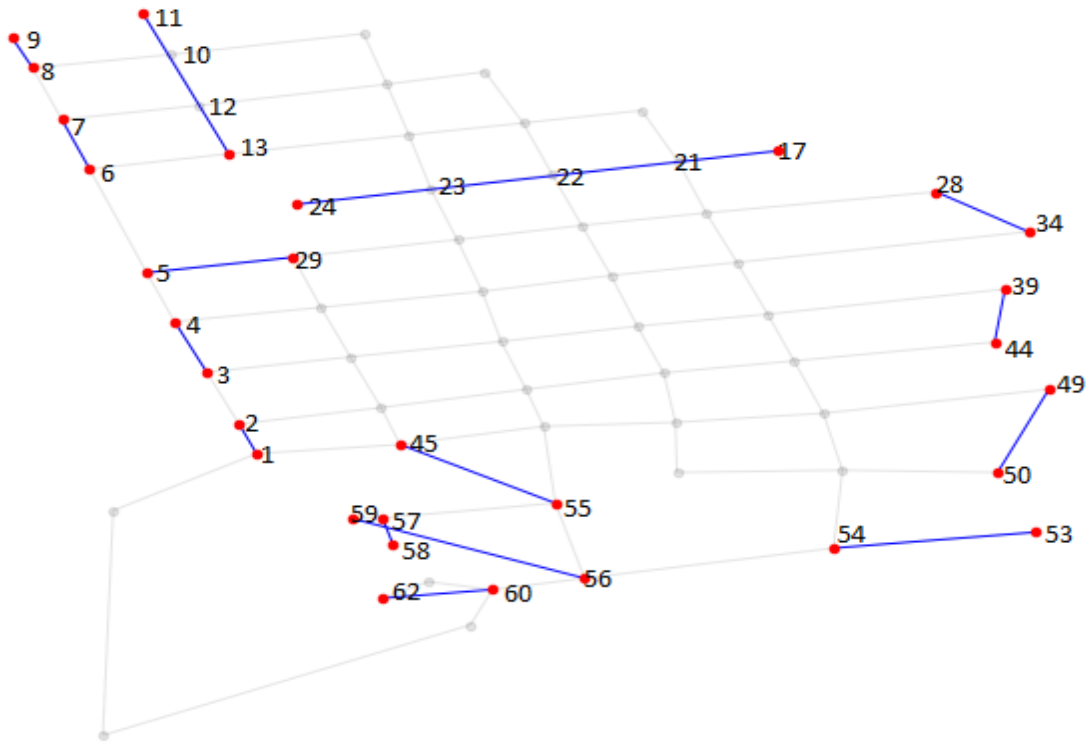
Pares de vértices de grau ímpar	
('vertice28', 'vertice34')	('vertice39', 'vertice44')
('vertice45', 'vertice55')	('vertice17', 'vertice24')
('vertice3', 'vertice4')	('vertice56', 'vertice59')
('vertice29', 'vertice5'	('vertice6', 'vertice7')
('vertice11', 'vertice13')	('vertice57', 'vertice58')
('vertice60', 'vertice62')	('vertice1', 'vertice2')
('vertice49', 'vertice50')	('vertice53', 'vertice54')
('vertice8', 'vertice9')	

Fonte: Autoria própria (2019).

Como pode ser observado na Figura 6 e no Quadro 3 o novo circuito gerado, possui um total de 108 arestas, mas ele possui alguns problemas, desse total de 108 arestas, algumas não são adjacentes no grafo original e não é possível percorrer de um vértice a outro em linha reta, como no caso dos pares de vértices 50-49, 55-45, 44-39, 34-28, 56-59 e 60-62, sendo que o percurso realizado efetivamente passa por mais de uma aresta do grafo real, como aquele apresentado na Figura 6.

O deslocamento entre os pares de vértices 11-13 e 17-24 também percorrem mais de uma aresta do grafo real como pode ser observado na Figura 6 e no Quadro 4. Sendo assim, o comprimento do circuito correto gerado, com base no grafo original é igual a 123 arestas.

Figura 6 - Grafo original com os vértices de grau ímpar, e as arestas adicionadas.



Fonte: Autoria própria, (2019).

Quadro 4 – Vértices percorridas no grafo original.

Pares de vértices percorridas e quantidades de arestas percorridas	
Vértices	**Quantidade de arestas percorridas**
11-13	3
17-24	4
50-49	3
55-45	2
44-39	3
34-28	3
56-59	2
60-62	2
57-58	1
54-53	1
5-29	1
4-3	1
7-6	1
8-9	1
1-2	1

Fonte: Autoria própria (2019).

Através da aplicação do algoritmo, foram ainda obtidos alguns dados estatísticos com relação ao grafo original, o mesmo contava com uma distância total de 9417.43 metros (soma do percurso de todas as arestas), por possuir vértices de grau ímpar fez-se necessário a adição de arestas ao mesmo, ocasionando o aumento do percurso total para 12518.19 metros (soma do percurso de todas as arestas do grafo mais o percurso total das arestas adicionadas), o que representa um aumento de 32,93% em relação a soma do tamanho de todas as arestas.

Logo o grafo original possuía 93 arestas e 64 vértices, com a criação do novo grafo euleriano o mesmo passou a ter 123 arestas com o mesmo número de vértices. Dos 65 vértices existentes, com a rota proposta ocorrerá que no percurso será passado por 21 vértices uma única vez, duas vezes por 30 vértices e três vezes por 14 vértices, totalizando 123, como apresentado no Quadro 5. Desta forma, 30

arestas são percorridas duas vezes e 63 arestas apenas uma vez, totalizando o percurso de 123 arestas, como apresentado no Quadro 6.

Quadro 5 – Número de vezes que cada vértice foi percorrido.

Nº de visitas	Nº de vértices
1	21
2	30
3	14

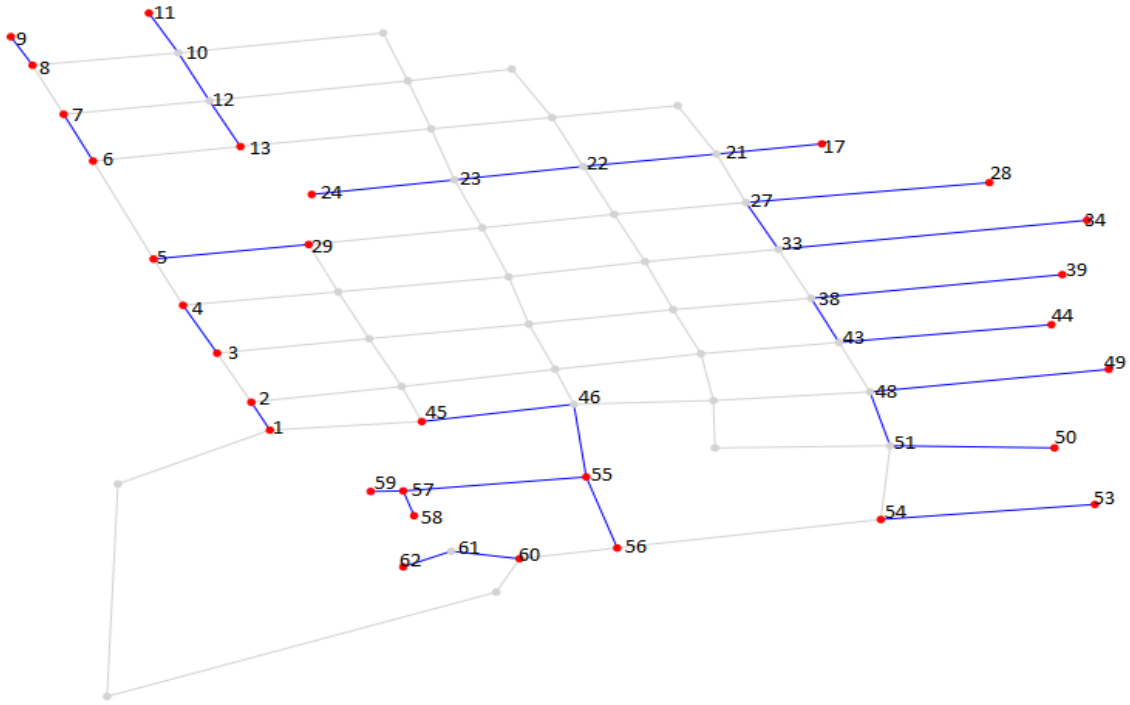
Fonte: Autoria própria, (2019).

Quadro 6 – Número de vezes que cada aresta foi visitada.

Nº de visitas	Nº de arestas
1	63
2	30

Fonte: Autoria própria (2019).

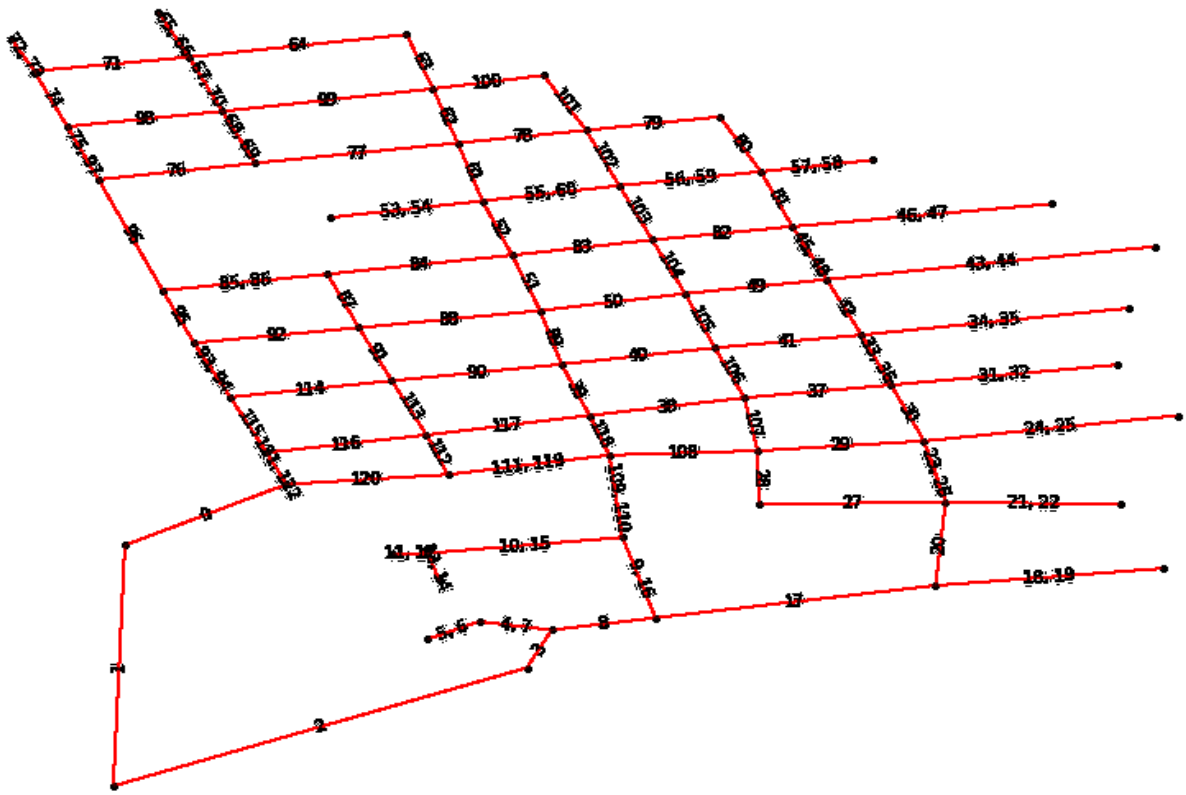
Figura 7 – Grafo com arestas adicionadas aos vértices de grau ímpar



Fonte: Autoria própria, (2019).

Pode-se observar ainda através da Figura 7, que as linhas em cinza presentam as arestas que são percorridas apenas uma única vez, já as arestas em azul são percorridas 2 vezes (que representam a menor distância possível), entre os vértices de grau ímpar que aparecem em vermelho.Na Figura 8, é apresentado a rota obtida pelos algoritmos utilizados e, portanto, proposta neste trabalho com a sequência em que os vértices devem ser percorridos, considerando o vértice 1 como ponto de partida. Os números separados por virgulas representam as arestas que são percorridas mais de 1 vez.

Figura 8 - Sequência de percurso dos vértices para a rota proposta



Fonte: Autoria própria (2019).

Pode-se observar com isto, que a rota aplicada a situação real, como se encontra mostrada na Figura 8, pode não ser interessante para o motorista do caminhão da coleta de resíduos, pois a mesma possui muitos cruzamentos e retornos em uma mesma rua, com poucas ruas sendo percorridas de forma sequencial de um extremo a outro do grafo.

Além disso, este percurso não pode ser considerado a rota ótima, primeiramente, pois foi escolhido de forma aleatória que o ponto de partida da rota de coleta seria o vértice 1 e, devido ao elevado número de cruzamentos apresentado. Para garantir a rota ótima seria necessária uma análise mais detalhada do presente estudo. Também seria interessante ter à disposição a rota efetuada atualmente na coleta para ter-se um parâmetro de comparação.

5 CONCLUSÃO

Neste trabalho fez-se o estudo da rota de coleta de resíduos orgânicos do bairro da Cango da cidade Francisco Beltrão-PR, visando a proposta de uma rota com menor percurso possível. Como o grafo gerado a partir do mapa possuía alguns vértices de grau ímpar foi necessária, fazer a adição de algumas arestas conectando pares de vértices de grau ímpar, e com o auxílio do algoritmo de Dijkstra, fez-se a soma das distâncias percorridas por estas arestas, e obteve-se o menor percurso possível. A adição destas arestas para tornar o grafo euleriano, só é possivel aplicando-se o método proposto por Edmonds e Johnson(1973).

Devido à presença de vértices de grau impar na rota proposta para a coleta do resíduo urbano orgânico no bairro da Cango em Francisco Beltrão, deve-se percorrer algumas arestas (ruas) mais de uma vez, sendo que isto acarreta um percurso que representa 32,93% do valor da soma das distâncias de todas as ruas.

Com relação ao algoritmo apresentado, podemos dizer que este obteve uma solução aproximada, para o percurso em estudo, não sendo possível, com o estudo realizado, que esta seja a rota de menor percurso, ou seja, a solução ótima. Para obter a solução ótima é necessária uma análise mais detalhada do problema analisado.

Sendo assim a efetiva aplicação da rota proposta, se tornaria viável se fosse implantada em um sistema de posicionamento global (GPS), onde a mesma seria inserida no sistema, não necessitando da memorização por parte do motorista do caminhão, e não ficaria restrita apenas a uma pessoa que saberia os caminhos a serem seguidos.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TECNICAS. **NBR. 10.004 - RESÍDUOS SÓLIDOS - CLASSIFICAÇÃO**: Resíduos Sólidos - Classificação. 2 ed. Rio de Janeiro: ABNT, 2004. 71 p.

BARROS, E. A. R.; PAMBOUKIAN, S. V. D.; ZAMBONI, L. C.; Algoritmo de dijkstra: apoio didático e multidisciplinar na implementação, simulação e utilização computacional.In: **International Conference on Engineering and Computer Education**, São Paulo,2007.

CARDOSO, R. P.; AVILA, S. L.; Matemática aplicada na busca do ótimo,**Seminário de pesquisa extensão e inovação do IFSC**,Florianopolis,2013.

CIDADES.**Página Institucional**. Disponível em: <https://cidades.ibge.gov.br/brasil/pr/francisco-beltrao/panorama>, Acesso em: 19 abr. 2019.

COLAB. Disponível em < https://colab.research.google.com/drive/1ydSUcSDPqwV4RdIlVuT36_VW_CnUJYp#scrollTo=BW7I8JfXJS3p >. Acesso em:29 de out. 2019.

CORRAR, Luiz J; THEÓFILO, Carlos R et. All; **Pesquisa Operacional para Decisão em Contabilidade e Administração.** São Paulo, Atlas, 2004.

EDMONDS, J.,JOHNSON,E.L. **Matching, euler tours and the chinese postman** North-Holland Publishing Company, Mathematical Programming 5,p.88-124.1973

FILHO, M. G., JUNQUEIRA, R. D. A. R. **Problema do Carteiro Chinês: escolha de métodos de solução e análise de tempos computacionais,** Produção, v. 16, n. 3, São Carlos,p. 538-551, Set. 2006.

FREITAS, F. G. D., MAIA, C. L. B., CAMPOS, G. A. L., SOUZA, J. T. D.**Otimização em Teste de *Software* com Aplicação de Metaheurísticas.** Revista de Sistemas de Informação da FSMA. Visconde de Araújo – RJ, n. 5, pp. 3-13.2010.

GOLDBARG M. C. et. al, **OTIMIZAÇÃO COMBINATÓRIA E META-HEURISTICAS: Algoritmos e Aplicações**, 1. Ed. Rio de Janeiro: p.48,Elsevier, 2016.

GOLDBARG, M. C.,LUNA. L. P. H.**Otimização combinatória e programação linear: modelos e algoritmos.**Elsevier.2.ed.Rio de Janeiro, 2005.

GOLVEIA,N.; Resíduos sólidos urbanos: impactos socioambientais e perspectiva de manejo sustentável com inclusão social, **Ciência & Saúde Coletiva**,São Paulo, p.1503-1510, 2012.

JUNIOR,A. F. **MÉTODO PARA OTIMIZAÇÃO DE ROTAS NA AGRICULTURA,**(Dissertação de mestrado)-Universidade de São Carlos,São Carlos ,2019.

KONOWALENKO, Flávia. **Problema do carteiro chinês não-orientado e misto para a otimização de rotas na cidade de Irati/PR.** Dissertação (Mestrado).

Programa de Pós-Graduação em Métodos Numéricos em Engenharia – Área: Programação Matemática, UFPR, Curitiba, 2012.

MEDEIROS, G. F., KRIPKA, M.**ALGUMAS APLICAÇÕES DE MÉTODOS HEURÍSTICOS NA OTIMIZAÇÃO DE ESTRUTURAS,** Revista CIATEC, vol.4 , 2012.

MIURA, M. **Resolução de um Problema de Roteamento de Veículos em uma Empresa Transportadora.** Escola Politécnica da Universidade de São Paulo, 2003.

MORO, M. F. **O PROBLEMA DO CARTEIRO CHINÊS APLICADO NA OTIMIZAÇÃO DE ROTAS USADAS NA COLETA DE LIXO RECICLÁVEL: UM ESTUDO DE CASO,** (Trabalho de conclusão de Curso)**,** UTFPR(Universidade Tecnologica Federal do Parana), Medianeira,2014.

MORO, M. F., ANDRADE, D. F. D., SANTOS, B. M. D., NETO, C. R. P., BATTISTI, J. D. F. **O problema do carteiro chinês aplicado na otimização das rotas de coleta de resíduos recicláveis: um estudo de caso,** Tecno-Lógica*,* Santa Cruz do Sul, v. 22, n. 2, p. 128-135, dez. 2018.

NETO, H. C., MARQUES, C. C., ARAÚJO, PAULO, G. C. D., GONÇALVES, W. P., MAIA R., BARBOS, E. A., **CARACTERIZAÇÃO DE RESÍDUOS SÓLIDOS ORGÂNICOS PRODUZIDOS NO RESTAURANTE UNIVERSITÁRIO DE UMA INSTITUIÇÃO PÚBLICA (ESTUDO DE CASO)**,XVVII Encontro Nacional de engenharia de produção A Energia que move a produção: um dialogo sobre integração, projeto e sustentabilidade,Foz do Iguaçu,2007.

ROSEN, K.H. **Matemática discreta e suas aplicações.**6º edição. Porto Alegre:AMGH,2010.

SILVA, D. F. D.; SANCHES, A. L. **Aplicação Conjunta do Método de Dijkstra e Otimização Combinatória para Solução do Problema do Caixeiro Viajante.** In: SEGET – Simpósio de excelência em gestão e tecnologia, 2004,Seget. P.1 – 9,Resende - RJ,2004.

SILVA, C. C. D. **ANÁLISE DE ESTABILIDADE DE UM TALUDE DA CAVA DE ALEGRIA UTILIZANDO ABORDAGEMPROBABILÍSTICA,**(Dissertação de mestrado) – Universidade Federal de Ouro Preto,Ouro Preto – MG,2015.

SANTOS ,H. R., JUNIOR, R. A. R. ***Aplicação do algoritmo de Dijkstra para o problema de roteamento da frota de táxis partindo de um ponto fixo***, Instituto Federal Fluminense, Rio de Janeiro,2010.

XAVIER, R. S., LISBOA, A. C., VIEIRA, D. A. G.**HEURÍSTICA PARA MODELAGEM E MINIMIZAÇÃO DO CONSUMO DE COMBUSTÍVEL PARA ROTAS DE COLETA DE LIXO.** 62º ENCONTRO DA SOCIEDADE BRASILEIRA DE PESQUISA OPERACIONAL, Bento Gonçalves-RS,2010.

ANEXO A - Problema do carteiro chinês resolvido pelo algoritmo de Dijkstra implementado em Python

```
import itertools

import copy

import networkx as nx

import pandas as pd

import matplotlib.pyplot as plt

# Grab edge list data hosted on Gist

edgelist=pd.read_csv('https://raw.githubusercontent.com/alceuc/Alceuc/master/rota.csv')

edgelist.head(94)

# Grab node list data hosted on Gist

nodelist = pd.read_csv('https://raw.githubusercontent.com/alceuc/Alceuc/master/latitude%20e%20longitude.csv')

nodelist.head(64)

# Create empty graph

g = nx.Graph()

# Add edges and edge attributes

for i, elrow in edgelist.iterrows():

        g.add_edge(elrow[0], elrow[1], **elrow[2:].to_dict())

        print(elrow[0], elrow[1])

# Edge list example

print(elrow[0]) # node1

print(elrow[1]) # node2

print(elrow[2:].to_dict()) # edge attribute dict

# Add node attributes

for i, nlrow in nodelist.iterrows():

   # g.node[nlrow['id']] = nlrow[1:].to_dict()  # deprecated after NX 1.11
```

```
    nx.set_node_attributes(g, {nlrow['id']:  nlrow[1:].to_dict()})
# Node list example
print(nlrow)
# Preview first 5 edges
# g.edges(data=True)[0:9]  # deprecated after NX 1.11
list(g.edges(data=True))[0:5]
# Preview first 10 nodes
# g.nodes(data=True)[0:10]  # deprecated after NX 1.11
list(g.nodes(data=True))[0:64]
print('# of edges: {}'.format(g.number_of_edges()))
print('# of nodes: {}'.format(g.number_of_nodes()))
g.nodes(data=True)
# Define node positions data structure (dict) for plotting
node_positions = {node[0]: (node[1]['X'], -node[1]['Y']) for node in
g.nodes(data=True)}
# Preview of node_positions with a bit of hack (there is no head/slice method for
dictionaries).
dict(list(node_positions.items())[0:64])
# Define data structure (list) of edge colors for plotting
# edge_colors = [e[2]['color'] for e in g.edges(data=True)]  # deprecated after NX 1.11
edge_colors = [e[2]['color'] for e in list(g.edges(data=True))]
# Preview first 10
edge_colors[0:10]
plt.figure(figsize=(8, 6))
nx.draw(g, pos=node_positions, edge_color=edge_colors, node_size=10,
node_color='black')
plt.title('Graphic representation of Cango neighborhood', size=15)
plt.show()
```

```
# Calculate list of nodes with odd degree

# nodes_odd_degree = [v for v, d in g.degree_iter() if d % 2 == 1]  # deprecated after NX 1.11

nodes_odd_degree = [v for v, d in g.degree() if d % 2 == 1]

# Preview

nodes_odd_degree[0:94]

# Counts

print('Number of nodes of odd degree: {}'.format(len(nodes_odd_degree)))

print('Number of total nodes: {}'.format(len(g.nodes())))

# Compute all pairs of odd nodes. in a list of tuples

odd_node_pairs = list(itertools.combinations(nodes_odd_degree, 2))

# Preview pairs of odd degree nodes

odd_node_pairs[0:94]

# Counts

print('Number of pairs: {}'.format(len(odd_node_pairs)))

def get_shortest_paths_distances(graph, pairs, edge_weight_name):

    """Compute shortest distance between each pair of nodes in a graph.  Return a dictionary keyed on node pairs (tuples)."""

    distances = {}

    for pair in pairs:

        distances[pair] = nx.dijkstra_path_length(graph, pair[0], pair[1],
        weight=edge_weight_name)

    return distances

# Compute shortest paths.  Return a dictionary with node pairs keys and a single value equal to shortest path distance.

odd_node_pairs_shortest_paths = get_shortest_paths_distances(g, odd_node_pairs, 'distance')

# Preview with a bit of hack (there is no head/slice method for dictionaries).

dict(list(odd_node_pairs_shortest_paths.items())[0:95])
```

```
def create_complete_graph(pair_weights, flip_weights=True):
    """Create a completely connected graph using a list of vertex pairs and the shortest path distances between them
    Parameters:
        pair_weights: list[tuple] from the output of get_shortest_paths_distances
        flip_weights: Boolean. Should we negate the edge attribute in pair_weights?  """
    g = nx.Graph()
    for k, v in pair_weights.items():
        wt_i = -v if flip_weights else v
            # g.add_edge(k[0], k[1], {'distance': v, 'weight': wt_i})  # deprecated after NX 1.11
            g.add_edge(k[0], k[1], **{'distance': v, 'weight': wt_i})
    return g

# Generate the complete graph

g_odd_complete = create_complete_graph(odd_node_pairs_shortest_paths, flip_weights=True)

# Counts

print('Number of nodes: {}'.format(len(g_odd_complete.nodes())))

print('Number of edges: {}'.format(len(g_odd_complete.edges())))

# Plot the complete graph of odd-degree nodes

plt.figure(figsize=(8, 6))

pos_random = nx.random_layout(g_odd_complete)

nx.draw_networkx_nodes(g_odd_complete, node_positions, node_size=20, node_color="red")

nx.draw_networkx_edges(g_odd_complete, node_positions, alpha=0.5)

plt.axis('on')

plt.title('Grafo completo com vertices de grau ímpar')

plt.show()
```

```
# Compute min weight matching.

# Note: max_weight_matching uses the 'weight' attribute by default as the attribute to maximize.

odd_matching_dupes = nx.algorithms.max_weight_matching(g_odd_complete, True)

print('Number of edges in matching: {}'.format(len(odd_matching_dupes)))

# Preview of matching with dupes

odd_matching_dupes

# Convert matching to list of deduped tuples

odd_matching = list(pd.unique([tuple(sorted([k, v])) for k, v in odd_matching_dupes]))

# Counts

print('Number of edges in matching (deduped): {}'.format(len(odd_matching)))

# Preview of deduped matching

odd_matching

plt.figure(figsize=(8, 6))

# Plot the complete graph of odd-degree nodes

nx.draw(g_odd_complete, pos=node_positions, node_size=20, alpha=0.1)

# Create a new graph to overlay on g_odd_complete with just the edges from the min weight matching

g_odd_complete_min_edges = nx.Graph(odd_matching)

nx.draw(g_odd_complete_min_edges, pos=node_positions, node_size=20, edge_color='blue', node_color='red'

plt.title('Min Weight Matching on Complete Graph')

plt.show()

plt.figure(figsize=(8, 6))

# Plot the original trail map graph

nx.draw(g, pos=node_positions, node_size=20, alpha=0.1, node_color='black')

# Plot graph to overlay with just the edges from the min weight matching
```

```
nx.draw(g_odd_complete_min_edges, pos=node_positions, node_size=20, alpha=1,
node_color='red', edge_color='blue')

plt.title('Arestas adicionadas ao grafo original')

plt.show()

def add_augmenting_path_to_graph(graph, min_weight_pairs):

    """Add the min weight matching edges to the original graph

    Parameters:

        graph: NetworkX graph (original graph from trailmap)

        min_weight_pairs: list[tuples] of node pairs from min weight matching

    Returns:

        augmented NetworkX graph """

    # We need to make the augmented graph a MultiGraph so we can add parallel
edges

    graph_aug = nx.MultiGraph(graph.copy())

    for pair in min_weight_pairs:

        graph_aug.add_edge(pair[0],

                           pair[1],

                           **{'distance': nx.dijkstra_path_length(graph, pair[0], pair[1]), 'trail':
'augmented'}

                           # attr_dict={'distance': nx.dijkstra_path_length(graph, pair[0],
pair[1]),

                                # 'trail': 'augmented'}  # deprecated after 1.11

                          )

    return graph_aug

# Create augmented graph: add the min weight matching edges to g

g_aug = add_augmenting_path_to_graph(g, odd_matching)

# Counts

print('Number of edges in original graph: {}'.format(len(g.edges())))
```

```
print('Number of edges in augmented graph: {}'.format(len(g_aug.edges())))

# pd.value_counts(g_aug.degree())  # deprecated after NX 1.11

pd.value_counts([e[1] for e in g_aug.degree()])

naive_euler_circuit = list(nx.eulerian_circuit(g_aug, source='vertice1'))

print('Length of eulerian circuit: {}'.format(len(naive_euler_circuit)))

# Preview naive Euler circuit

naive_euler_circuit[0:94]

def create_eulerian_circuit(graph_augmented, graph_original, starting_node=None):

    """Create the eulerian path using only edges from the original graph."""

    euler_circuit = []

    naive_circuit = list(nx.eulerian_circuit(graph_augmented, source=starting_node))

    for edge in naive_circuit:

        edge_data = graph_augmented.get_edge_data(edge[0], edge[1])

        if edge_data[0]['trail'] != 'augmented':

            # If `edge` exists in original graph, grab the edge attributes and add to
eulerian circuit.

            edge_att = graph_original[edge[0]][edge[1]]

            euler_circuit.append((edge[0], edge[1], edge_att))

        else:

            aug_path = nx.shortest_path(graph_original, edge[0], edge[1],
weight='distance')

            aug_path_pairs = list(zip(aug_path[:-1], aug_path[1:]))

            print('Filling in edges for augmented edge: {}'.format(edge))

            print('Augmenting path: {}'.format(' => '.join(aug_path)))

            print('Augmenting path pairs: {}\n'.format(aug_path_pairs))

            # If `edge` does not exist in original graph, find the shortest path between its
nodes and

            #  add the edge attributes for each link in the shortest path.
```

```
            for edge_aug in aug_path_pairs:
                edge_aug_att = graph_original[edge_aug[0]][edge_aug[1]]
                euler_circuit.append((edge_aug[0], edge_aug[1], edge_aug_att))
    return euler_circuit
# Create the Eulerian circuit
euler_circuit = create_eulerian_circuit(g_aug, g, 'vertice1')
print('Length of Eulerian circuit: {}'.format(len(euler_circuit)))
# Preview first 20 directions of CPP solution
for i, edge in enumerate(euler_circuit[0:20]):
    print(i, edge)
# Computing some stats
total_mileage_of_circuit = sum([edge[2]['distance'] for edge in euler_circuit])
total_mileage_on_orig_trail_map = sum(nx.get_edge_attributes(g,
'distance').values())
_vcn = pd.value_counts(pd.value_counts([(e[0]) for e in euler_circuit]), sort=False)
node_visits = pd.DataFrame({'n_visits': _vcn.index, 'n_nodes': _vcn.values})
_vce = pd.value_counts(pd.value_counts([sorted(e)[0] + sorted(e)[1] for e in
nx.MultiDiGraph(euler_circuit).edges()]))
edge_visits = pd.DataFrame({'n_visits': _vce.index, 'n_edges': _vce.values})
# Printing stats
print('Mileage of circuit: {0:.2f}'.format(total_mileage_of_circuit))
print('Mileage on original trail map: {0:.2f}'.format(total_mileage_on_orig_trail_map))
print('Mileage retracing edges: {0:.2f}'.format(total_mileage_of_circuit-
total_mileage_on_orig_trail_map))
print('Percent of mileage retraced: {0:.2f}%\n'.format((1-
total_mileage_of_circuit/total_mileage_on_orig_trail_map)*-100))
print('Number of edges in circuit: {}'.format(len(euler_circuit)))
print('Number of edges in original graph: {}'.format(len(g.edges())))
```

```
print('Number of nodes in original graph: {}\n'.format(len(g.nodes())))

print('Number of edges traversed more than once: {}\n'.format(len(euler_circuit)-
len(g.edges())))

print('Number of times visiting each node:')

print(node_visits.to_string(index=False))

print('\nNumber of times visiting each edge:')

print(edge_visits.to_string(index=False))

def create_cpp_edgelist(euler_circuit):
    """
    Create the edgelist without parallel edge for the visualization
    Combine duplicate edges and keep track of their sequence and # of walks
    Parameters:
        euler_circuit: list[tuple] from create_eulerian_circuit
    """
    cpp_edgelist = {}
    for i, e in enumerate(euler_circuit):
        edge = frozenset([e[0], e[1]])
        if edge in cpp_edgelist:
            cpp_edgelist[edge][2]['sequence'] += ', ' + str(i)
            cpp_edgelist[edge][2]['visits'] += 1
        else:
            cpp_edgelist[edge] = e
            cpp_edgelist[edge][2]['sequence'] = str(i)
            cpp_edgelist[edge][2]['visits'] =
    return list(cpp_edgelist.values())
cpp_edgelist = create_cpp_edgelist(euler_circuit)

print('Number of edges in CPP edge list: {}'.format(len(cpp_edgelist)))
```

```
# Preview CPP plot-friendly edge list

cpp_edgelist[0:3]

# Create CPP solution graph

g_cpp = nx.Graph(cpp_edgelist)

plt.figure(figsize=(14, 10))

visit_colors = {1:'lightgray', 2:'blue'}

edge_colors = [visit_colors[e[2]['visits']] for e in g_cpp.edges(data=True)]

node_colors = ['red'  if node in nodes_odd_degree else 'lightgray' for node in
g_cpp.nodes()]

nx.draw_networkx(g_cpp, pos=node_positions, node_size=20,
node_color=node_colors, edge_color=edge_colors, with_labels=False)

plt.axis('off')

plt.show()

plt.figure(figsize=(14, 10))

edge_colors = [e[2]['color'] for e in g_cpp.edges(data=True)]

nx.draw_networkx(g_cpp, pos=node_positions, node_size=10, node_color='black',
edge_color=edge_colors, with_labels=False, alpha=0.5)

bbox = {'ec':[1,1,1,0], 'fc':[1,1,1,0]}  # hack to label edges over line (rather than
breaking up line)

edge_labels = nx.get_edge_attributes(g_cpp, 'sequence')

nx.draw_networkx_edge_labels(g_cpp, pos=node_positions,
edge_labels=edge_labels, bbox=bbox, font_size=10)

plt.axis('off')

plt.show()
```

Fonte:Colab,(2019).

APÊNDICE A - Rotas com distâncias entre os vértices.

Rotas com as distâncias:

node1,node2,trail,distance,color,estimate

vertice1,vertice2,rua,44.54,red,1

vertice1,vertice45,rua,98.64,red,1

vertice2,vertice3,rua,80.5,red,1

vertice2,vertice40,rua,96.0,red,1

vertice3,vertice35,rua,96.5,red,1

vertice3,vertice4,rua,88.0,red,1

vertice4,vertice5,rua,79.5,red,1

vertice4,vertice30,rua,96.5,red,1

vertice5,vertice6,rua,133.3,red,1

vertice5,vertice29,rua,96.5,red,1

vertice6,vertice7,rua,78.3,red,1

vertice6,vertice13,rua,95.0,red,1

vertice7,vertice8,rua,80.0,red,1

vertice7,vertice12,rua,95.0,red,1

vertice8,vertice9,rua,49.0,red,1

vertice8,vertice10,rua,94.0,red,1

vertice19,vertice18,rua,78.3,red,1

vertice16,vertice15,rua,80.11,red,1

vertice15,vertice12,rua,136.38,red,1

vertice15,vertice18,rua,69.0,red,1

vertice15,vertice14,rua,78.41,red,1

vertice14,vertice19,rua,73.84,red,1

vertice19,vertice20,rua,86.0,red,1

vertice19,vertice22,rua,81.3,red,1

vertice22,vertice26,rua,80.0,red,1

vertice21,vertice17,rua,87.0,red,1

vertice21,vertice27,rua,80.0,red,1

vertice27,vertice26,rua,86.0,red,1

vertice27,vertice28,rua,203.0,red,1

vertice27,vertice33,rua,79.5,red,1

vertice26,vertice32,rua,79.5,red,1

vertice26,vertice25,rua,84.42,red,1

vertice32,vertice33,rua,86.0,red,1

vertice33,vertice34,rua,240.52,red,1

vertice32,vertice31,rua,88.68,red,1

vertice32,vertice37,rua,80.0,red,1

vertice31,vertice36,rua,80.1,red,1

vertice36,vertice37,rua,93.61,red,1

vertice23,vertice24,rua,105.54,red,1

vertice23,vertice14,rua,81.42,red,1

vertice14,vertice13,rua,131.54,red,1

vertice13,vertice12,rua,78.3,red,1

vertice12,vertice10,rua,80.0,red,1

vertice10,vertice11,rua,105.6,red,1

vertice10,vertice16,rua,153.39,red,1

vertice37,vertice38,rua,86.0,red,1

vertice38,vertice39,rua,199.9,red,1

vertice37,vertice42,rua,80.5,red,1

vertice38,vertice43,rua,80.5,red,1

vertice36,vertice41,rua,80.62,red,1

vertice43,vertice44,rua,153.6,red,1

vertice42,vertice43,rua,86.0,red,1

vertice41,vertice42,rua,98.57,red,1

vertice46,vertice41,rua,44.4,red,1

vertice47,vertice42,rua,66.7,red,1

vertice48,vertice43,rua,80.5,red,1

vertice45,vertice46,rua,98.65,red,1

vertice46,vertice47,rua,97.52,red,1

vertice47,vertice48,rua,87.06,red,1

vertice48,vertice49,rua,158.6,red,1

vertice47,vertice52,rua,71.86,red,1

vertice52,vertice51,rua,111.0,red,1

vertice48,vertice51,rua,84.73,red,1

vertice51,vertice50,rua,147.0,red,1

vertice51,vertice54,rua,120.14,red,1

vertice54,vertice53,rua,118.0,red,1

vertice46,vertice55,rua,117.4,red,1

vertice45,vertice40,rua,49.36,red,1

vertice40,vertice35,rua,80.5,red,1

vertice40,vertice41,rua,103.56,red,1

vertice35,vertice36,rua,105.63,red,1

vertice35,vertice30,rua,80.1,red,1

vertice30,vertice29,rua,79.5,red,1

vertice30,vertice31,rua,110.56,red,1

vertice29,vertice25,rua,114.92,red,1

vertice25,vertice23,rua,80.1,red,1

vertice55,vertice56,rua,117.8,red,1

vertice55,vertice57,rua,122.0,red,1

vertice57,vertice58,rua,67.2,red,1

vertice57,vertice59,rua,30.0,red,1

vertice56,vertice60,rua,67.3,red,1

vertice60,vertice61,rua,65.0,red,1

vertice60,vertice63,rua,61.7,red,1

vertice61,vertice62,rua,38.1,red,1

vertice63,vertice64,rua,385.87,red,1

vertice64,vertice65,rua,391.48,red,1

vertice65,vertice1,rua,149.72,red,1

vertice31,vertice25,rua,79.60,red,1

vertice23,vertice22,rua,80.48,red,1

vertice22,vertice21,rua,86.00,red,1

vertice21,vertice20,rua,81.30,red,1

vertice33,vertice38,rua,80.00,red,1

vertice54,vertice56,rua,171.66,red,1

APÊNDICE B - Latitude e longitude dos nós ou vértices do grafo original

Latitude e longitude

id,X,Y

vertice1,6720,5558

vertice2,6708,5506

vertice3,6686,5415

vertice4,6664,5326

vertice5,6645,5240

vertice6,6606,5058

vertice7,6587,4971

vertice8,6567,4880

vertice9,6553,4827

vertice10,6661,4857

vertice11,6642,4783

vertice12,6681,4946

vertice13,6701,5031

vertice14,6824,4998

vertice15,6809,4909

vertice16,6793,4820

vertice17,7076,5026

vertice18,6876,4887

vertice19,6902,4977

vertice20,6983,4955

vertice21,7008,5045

vertice22,6922,5068

vertice23,6839,5093

vertice24,6747,5120

vertice25,6857,5182

vertice26,6942,5157

vertice27,7027,5135

vertice28,7184,5098

vertice29,6745,5213

vertice30,6764,5301

vertice31,6874,5273

vertice32,6962,5245

vertice33,7048,5222

vertice34,7247,5168

vertice35,6784,5388

vertice36,6887,5361

vertice37,6980,5334

vertice38,7069,5313

vertice39,7231,5269

vertice40,6805,5477

vertice41,6904,5445

vertice42,6998,5416

vertice43,7087,5395

vertice44,7224,5362

vertice45,6818,5542

vertice46,6916,5510

vertice47,7006,5503

vertice48,7107,5487

vertice49,7261,5445

vertice50,7226,5591

vertice51,7120,5587

vertice52,7007,5591

vertice53,7252,5696

vertice54,7114,5724

vertice55,6924,5645

vertice56,6944,5777

vertice57,6806,5671

vertice58,6813,5717

vertice59,6785,5672

vertice60,6881,5797

vertice61,6837,5783

vertice62,6806,5812

vertice63,6866,5859

vertice64,6615,6053

vertice65,6622,5658