



UNIVERSIDADE ESTADUAL DO CEARÁ
CENTRO DE CIÊNCIAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO BASTOS VASCONCELOS

**O PROBLEMA DO CARTEIRO CHINÊS DIRIGIDO, NÃO DIRIGIDO E MISTO
PARA OTIMIZAÇÃO DE ROTAS COM VISUALIZAÇÃO GRÁFICA DA SOLUÇÃO**

FORTALEZA – CEARÁ

2017

RODRIGO BASTOS VASCONCELOS

O PROBLEMA DO CARTEIRO CHINÊS DIRIGIDO, NÃO DIRIGIDO E MISTO PARA
OTIMIZAÇÃO DE ROTAS COM VISUALIZAÇÃO GRÁFICA DA SOLUÇÃO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientador: Gerardo Valdisio Rodrigues Viana

Co-Orientador: Leonardo Sampaio Rocha

FORTALEZA – CEARÁ

2017

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Sistema de Bibliotecas

Vasconcelos, Rodrigo Bastos.

O Problema do Carteiro Chinês dirigido, não dirigido e misto para otimização de rotas com visualização gráfica da solução [recurso eletrônico] / Rodrigo Bastos Vasconcelos. - 2017.

1 CD-ROM: il.; 4 ¾ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico com 102 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

Dissertação (mestrado acadêmico) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2017.

Área de concentração: Otimização e Matemática Aplicada.

Orientação: Prof. Dr. Gerardo Valdisio Rodrigues Viana.

Coorientação: Prof. Dr. Leonardo Sampaio Rocha.

1. Problema do Carteiro Chinês. 2. PCC não dirigido. 3. PCC dirigido. 4. PCC misto. 5. Otimização. I. Título.

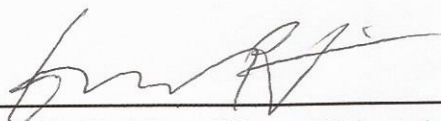
RODRIGO BASTOS VASCONCELOS

O PROBLEMA DO CARTEIRO CHINÊS DIRIGIDO, NÃO DIRIGIDO E MISTO PARA
OTIMIZAÇÃO DE ROTAS COM VISUALIZAÇÃO GRÁFICA DA SOLUÇÃO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em: 15 de Fevereiro de 2017

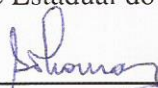
BANCA EXAMINADORA



Gerardo Valdisio Rodrigues Viana (Orientador)
Universidade Estadual do Ceará – UECE



Leonardo Sampaio Rocha (Co-Orientador)
Universidade Estadual do Ceará – UECE



Antônio Clécio Fontelles Thomaz
Universidade Estadual do Ceará - UECE



Plácido Rogério Pinheiro
Universidade de Fortaleza - UNIFOR

AGRADECIMENTOS

Quando Deus age na vida de uma pessoa, e ele age na vida de todos nós, ele faz maravilhas, mesmo que não tenhamos a percepção disso. Por isso, devo agradecer a Deus por todas as maravilhas que Ele fez em minha vida e por Ele ter me proporcionado todo o necessário para que eu conquistasse mais esta etapa da minha vida.

Venho agradecer também aos meus pais. Ambos, sempre serão grandes exemplos para mim, e a educação e o incentivo que sempre me deram foi fundamental para que eu conquistasse tudo que eu conquistei até hoje.

Meu Pai, você foi, ainda é, e sempre será uma grande inspiração para mim. Obrigado por ter me dito várias e várias vezes que o que uma pessoa faz, qualquer pessoa é capaz de fazer. E obrigado também por me tranquilizar sempre que eu preciso. Eu te amo!

Minha mãe, obrigado por você ser do jeito que você é. Uma heroína, é isso que eu acho que você é. Obrigado por tudo. Eu te amo!

À minha esposa, Lara, gostaria de agradecer imensamente por toda paciência que teve comigo. Mesmo nas horas mais difíceis, sempre estava ao meu lado me motivando e não me deixando desanimar. Obrigado meu amor por tudo.

Agradeço também ao meu professor orientador Valdisio. Pelas excelentes aulas que ele lecionou desde a graduação e pela paciência e atenção na orientação deste trabalho.

Não poderia esquecer de agradecer meu avô Airles. Obrigado vovô por toda a ajuda que o senhor sempre me deu. Obrigado por ser até hoje um exemplo para mim. Obrigado por ter me feito entender, através do seu exemplo, que um homem consegue tudo que quer quando tem disciplina. Por fim agradeço à minha vovozinha Célia, que sempre foi um anjinho de Deus em minha vida. Onde quer que você esteja, você estará para sempre no meu coração.

RESUMO

Os problemas de roteamento têm como objetivo determinar, em um grafo, um circuito de custo mínimo passando por todos os vértices ou por todas as arestas deste grafo, dependendo se o problema abordado está na classe de Problemas do Caixeiro Viajante (PCV) ou na classe de Problemas do Carteiro Chinês (PCC). Este trabalho discorre especificamente sobre os problemas contidos na classe de Problemas do Carteiro Chinês. Os problemas contidos nesta classe são NP-completos, deste modo, não existe algoritmo determinístico polinomial que encontre a solução exata para qualquer instância do problema. O problema tratado consiste em determinar um caminho mínimo que se inicie em algum vértice do grafo, passe por todas as arestas dele pelo menos uma vez e retorne ao vértice inicial. As variações deste problema se dividem basicamente em: Problema do Carteiro Chinês Dirigido (PCCD), Problema do Carteiro Chinês Não Dirigido (PCCND) e Problema do Carteiro Chinês Misto (PCCM), dependendo da natureza do grafo utilizado. Este trabalho formaliza uma maneira de calcular o percurso do carteiro chinês em cada uma destas variações, e fornece também uma visualização gráfica do grafo com uma animação do percurso realizado pelo carteiro. Além disso, o algoritmo desenvolvido é aplicado em um exemplo real do problema de coleta de lixo na cidade de Fortaleza.

Palavras-chave: Problema do Carteiro Chinês. PCC. PCC dirigido. PCC não dirigido. PCC misto. Coleta de Lixo

ABSTRACT

Routing problems are intended to determine, in a graph, a minimum cost circuit through all the vertices or all edges of this graph, depending on whether the problem addressed is in the class of Travelling Salesman Problems (TSP) or in the class of Chinese Postman Problems (CPP). This paper specifically discusses the problems contained in the Chinese Postman Problem class. The problems contained in this class are NP-complete, so there is no polynomial deterministic algorithm to find the exact solution to any instance of the problem. This problem is to determine a minimum path that begins at some vertex of the graph, go through all the edges of it at least once and return to the initial vertex. Variations of this problem are divided basically: directed CPP (DCPP), undirected CPP (UCPP) and mixed CPP (MCP), depending on the nature of the graph used. This work formalizes a way to calculate the route of the Chinese Postman in each of these variations, and also provides a graphical display with a graph animation of the path followed by the postman. Moreover, the developed algorithm is applied in a real example of the garbage collection problem in the city of Fortaleza.

Keywords: Chinese Postman Problem. CPP. Directed CPP. Undirected CPP. Mixed CPP. Garbage Collection

LISTA DE FIGURAS

Figura 1 – As 10 redes rodoviárias mais extensas do Mundo	17
Figura 2 – As sete pontes de Königsberg	21
Figura 3 – Grafo representativo das pontes de Königsberg	22
Figura 4 – Aresta e_1 que liga os vértices n_1 e n_2	23
Figura 5 – Arco a_1 que liga os vértices n_1 e n_2	23
Figura 6 – Links l_1 e l_2 que ligam os vértices n_1, n_2 e n_3	24
Figura 7 – Aresta (n_i, n_j)	24
Figura 8 – Arcos (n_i, n_j) e (n_j, n_i) contrariamente direcionados	24
Figura 9 – Exemplo de grafo valorado	24
Figura 10 – Subgrafo gerador do grafo da Figura 10	25
Figura 11 – Grafo não dirigido e f-conexo	27
Figura 12 – Grafo não dirigido e desconexo	27
Figura 13 – Grafo dirigido e fortemente conexo	28
Figura 14 – Grafo euleriano	28
Figura 15 – Grafo semi-euleriano	29
Figura 16 – Grafo euleriano utilizado para descrever o algoritmo de Hierholzer	31
Figura 17 – Primeira iteração do algoritmo de Hierholzer	31
Figura 18 – Segunda iteração do algoritmo de Hierholzer	32
Figura 19 – Resultado do algoritmo de Hierholzer	32
Figura 20 – Grafo com uma aresta que é uma ponte	33
Figura 21 – Grafo desconexo após retirada da aresta que era uma ponte	33
Figura 22 – Primeira iteração do algoritmo de Fleury	34
Figura 23 – Primeira iteração do algoritmo de Fleury Segunda iteração do algoritmo de Fleury	34
Figura 24 – Terceira iteração do algoritmo de Fleury	34
Figura 25 – Quarta iteração do algoritmo de Fleury	35
Figura 26 – Quinta iteração do algoritmo de Fleury	35
Figura 27 – Sexta iteração do algoritmo de Fleury	35
Figura 28 – Última iteração do algoritmo de Fleury	36
Figura 29 – Fluxograma para encontrar um circuito euleriano em um grafo	37
Figura 30 – Exemplo de Grafo Euleriano não ótimo	37

Figura 31 – Vértices associados de maneira não ótima	38
Figura 32 – Arestas duplicadas entre os vértices associados	38
Figura 33 – Vértices associados de maneira ótima	39
Figura 34 – Exemplos de Blossoms	40
Figura 35 – Possível emparelhamento máximo sobre os blossoms da Figura 34	41
Figura 36 – Exemplo de Grafo Não Dirigido	41
Figura 37 – Exemplo de Grafo Dirigido	42
Figura 38 – Exemplo de Grafo aplicado ao Problema de Transportes	43
Figura 39 – Exemplo de Grafo Misto	44
Figura 40 – Relaxação através da desigualdade triangular	46
Figura 41 – Exemplo de grafo com ciclo negativo destacado em vermelho	47
Figura 42 – Algoritmo que auxilia na escolha de uma abordagem adequada para a resolução do PCC	51
Figura 43 – Fluxograma básico da execução dos algoritmos propostos	56
Figura 44 – Grafo exemplo PCCND	58
Figura 45 – Grafo para exemplo de execução do PCCND com vértices ímpares selecionados	59
Figura 46 – Grafo para exemplo de execução do PCCND com vértices pares removidos	59
Figura 47 – Grafo para exemplo de execução do PCCND após adicionadas cópias de arestas	60
Figura 48 – Grafo euleriano para exemplo de execução do PCCND após adicionadas cópias de arestas	61
Figura 49 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCND	62
Figura 50 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCND	62
Figura 51 – Fluxograma da execução do PCCD	63
Figura 52 – Grafo para exemplo de execução do PCCD	65
Figura 53 – Grafo para exemplo de execução do PCCD com vértices desbalanceados selecionados	65
Figura 54 – Grafo para exemplo de execução do PCCD com vértices balanceados removidos	66
Figura 55 – Grafo para exemplo de execução do PCCD após adicionadas cópias de arcos	67
Figura 56 – Grafo euleriano para exemplo de execução do PCCD após adicionadas cópias de arcos	69

Figura 57 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCD	70
Figura 58 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCD	70
Figura 59 – Fluxograma da execução do PCCM	71
Figura 60 – Grafo para exemplo de execução do PCCM	71
Figura 61 – Grafo para exemplo de execução do PCCM após transformação em grafo dirigido	72
Figura 62 – Grafo para exemplo de execução do PCCM com vértices desbalanceados destacados	72
Figura 63 – Grafo para exemplo de execução do PCCM com vértices balanceados removidos	73
Figura 64 – Grafo para exemplo de execução do PCCM após adicionadas cópias de arcos	74
Figura 65 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCM	75
Figura 66 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCM	75
Figura 67 – Padronização dos nomes das variáveis do modelo de PLI criado	77
Figura 68 – Montagem de cada termo do modelo de PLI criado	78
Figura 69 – Ruas dispostas no mapa	81
Figura 70 – Representação do problema de coleta de lixo como um grafo	82

LISTA DE TABELAS

Tabela 0 – Formas de resolver o PCCD	43
Tabela 1 – Principais objetos CPLEX	76
Tabela 2 – Exemplos das principais operações do CPLEX	76
Tabela 3 – Ruas da cidade de Fortaleza a serem percorridas	82
Tabela 4 – Resultados obtidos na execução do exemplo	83

LISTA DE QUADROS

Quadro 1 – Composição dos Custos Logísticos do Brasil em 2006	16
Quadro 2 – Distâncias Mínimas exemplo execução PCCND	59
Quadro 3 – Resultado da execução do modelo de PLI 1 no exemplo de execução do PCCND	61
Quadro 4 – Informações da execução do exemplo de PCCND	63
Quadro 5 – Distâncias mínimas para o exemplo execução PCCD	66
Quadro 6 – Resultado da execução do modelo de PLI 2 no exemplo de execução do PCCD	68
Quadro 7 – Informações da execução do exemplo de PCCD	68
Quadro 8 – Distâncias mínimas para o exemplo execução PCCM	73
Quadro 9 – Execução do modelo de PLI 2 no exemplo do PCCM	74
Quadro 10 – Informações da execução do exemplo de PCCM	75

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo Blossom I	86
Algoritmo 2 – Algoritmo que adiciona o tipo de função objetivo	86
Algoritmo 3 – Algoritmo de Hierholzer	87
Algoritmo 4 – Algoritmo de Fleury	87
Algoritmo 5 – Algoritmo Blossom I	88
Algoritmo 6 – Algoritmo de Dijkstra	88
Algoritmo 7 – Algoritmo de Relaxação de Dijkstra	89
Algoritmo 8 – Inicialização do Algoritmo de Dijkstra	89
Algoritmo 9 – Algoritmo de Bellman-Ford	90
Algoritmo 10 – Algoritmo Básico resolução do PCC	91
Algoritmo 11 – Algoritmo para calcular o emparelhamento ótimo do grafo	91
Algoritmo 12 – Algoritmo para tornar um grafo euleriano	92
Algoritmo 13 – Algoritmo para resolução do PCCM	92
Algoritmo 14 – Checagem de ciclos negativos no Algoritmo de Bellman-Ford	93
Algoritmo 15 – Algoritmo para transformar um grafo misto em um grafo dirigido equi- valente	93
Algoritmo 16 – Algoritmo para gerar o modelo de PLI utilizando o solver CPLEX	94
Algoritmo 17 – Algoritmo para adicionar as variáveis ao modelo de PLI	94
Algoritmo 18 – Algoritmo que converte uma aresta em uma variável a ser adicionada ao modelo de PLI	95
Algoritmo 19 – Algoritmo que adiciona cada termo à função objetivo do modelo de PLI	95
Algoritmo 20 – Algoritmo que adiciona o tipo de função objetivo	95
Algoritmo 21 – Algoritmo que adiciona as restrições à modelagem matemático do PCC não dirigido	96
Algoritmo 22 – Algoritmo que adiciona as restrições à modelagem matemático do PCC dirigido	97
Algoritmo 23 – Algoritmo para encontrar um caminho aumentado em um grafo	98

SUMÁRIO

1	INTRODUÇÃO	16
2	OBJETIVOS	20
2.1	OBJETIVO GERAL	20
2.2	OBJETIVOS ESPECÍFICOS	20
2.3	ORGANIZAÇÃO DO TRABALHO	20
3	FUNDAMENTAÇÃO TEÓRICA	21
3.1	HISTÓRICO	21
3.2	DEFINIÇÕES	23
3.3	CIRCUITOS EULERIANOS	27
3.3.1	Algoritmos para encontrar Circuitos Eulerianos	30
3.3.1.1	Algoritmo de Hierholzer	30
3.3.1.2	Algoritmo de Fleury	31
3.4	PROBLEMA DO CARTEIRO CHINÊS	33
3.4.1	Emparelhamento	37
3.4.2	Tipos de Problemas do Carteiro Chinês	39
3.4.2.1	Problema do Carteiro Chinês Não Dirigido (PCCND)	40
3.4.2.2	Problema do Carteiro Chinês Dirigido (PCCD)	42
3.4.2.3	Problema do Carteiro Chinês Misto (PCCM)	43
3.5	ALGORITMOS DE CAMINHO MÍNIMO	45
3.5.1	Algoritmo de Dijkstra	45
3.5.2	Algoritmo de Bellman-Ford	46
4	TRABALHOS RELACIONADOS	48
4.1	PROBLEMA DO CARTEIRO CHINÊS COM SOLUÇÕES EXATAS	48
4.1.1	Problema do Carteiro Chinês: escolha de métodos de solução	48
4.1.2	O problema do carteiro chinês, algoritmos exatos e um ambiente MVI para análise de suas instâncias: sistema XNÊS	52
4.2	PROBLEMA DO CARTEIRO CHINÊS COM SOLUÇÕES APROXIMADAS	53
4.2.1	Aplicação de um algoritmo genético para o problema do carteiro chinês em uma situação real de cobertura de arcos	53

4.2.2	Otimização de rotas para a coleta do lixo doméstico: um tratamento GRASP do PCCM	54
4.2.2.1	Fase de Construção	54
4.2.2.2	Fase de Busca Local	54
4.2.3	Busca Tabu aplicada ao problema de roteirização de veículos	55
5	METODOLOGIA	56
5.1	PCC NÃO DIRIGIDO	57
5.1.1	Execução do Algoritmo para o PCCND	58
5.2	PCC DIRIGIDO	63
5.2.1	Execução do Algoritmo para o PCCD	64
5.3	PCC MISTO	69
5.3.1	Execução do Algoritmo para o PCCM	71
5.4	GERAÇÃO MODELAGEM MATEMÁTICA UTILIZANDO CPLEX	76
5.4.1	Adição das restrições aos modelos de PLI utilizando CPLEX	78
5.4.1.1	PCC não dirigido	78
5.4.1.2	PCC dirigido	78
5.4.1.3	PCC misto	79
6	RESULTADOS	80
6.1	EXEMPLO DE COLETA DE LIXO NA CIDADE DE FORTALEZA	80
7	CONCLUSÕES E TRABALHOS FUTUROS	84
7.1	CONTRIBUIÇÕES DO TRABALHO	84
7.2	LIMITAÇÕES	84
7.3	TRABALHOS FUTUROS	84
APÊNDICE		85
	APÊNDICE A – ALGORITMOS	86

1 INTRODUÇÃO

O problema de distribuir bens ou atender serviços em diversos pontos geográficos é um problema de logística que ganha muita importância nos dias de hoje. Cada vez se torna mais importante realizar estas entregas ou efetuar estes serviços nestes pontos com maior rapidez e com menor custo de tempo e dinheiro. Segundo (??) Existe uma tendência de migração dos mercados produtores conforme a vocação regional. Isso pode significar um distanciamento maior destes produtores dos mercados de consumo. Porém, por outro lado, o comércio eletrônico disponibiliza estes produtos e serviços a todo consumidor potencial, sem limite geográfico. Cada vez menos ele precisa sair de seu lugar na busca de produtos e serviços. Paradoxalmente, na mesma medida que os centros produtores se distanciaram do mercado consumidor, os produtos em si chegaram muito mais próximos ao seu destino final. Para a comodidade dos consumidores e para o lucro dos fornecedores, estes produtos devem chegar o mais rapidamente possível em seus destinos, e para que isso aconteça é necessário planejar uma estratégia de logística adequada para o seu problema.

Os custos logísticos representam um tipo de custo muito significativo dentro das cadeias produtivas e principalmente para as empresas, somando-se aos custos identificados nos estoques, inventário, embalagem, fluxo de informações, movimentação, aspectos legais, planejamento operacional, armazenagem, serviço ao cliente, suprimentos e planejamento estratégicos. Estima-se que o custo logístico em uma empresa possa equivaler a 19% do seu faturamento (??). A composição de todos estes custos logísticos do Brasil em 2006 chegava a um total de R\$ 222 bilhões, o equivalente a 12,6% do PIB (??) deste mesmo ano.

Quadro 1 – Composição dos Custos Logísticos do Brasil em 2006

Setor Logístico	Porcentagem
TRANSPORTE	7,5%
ESTOQUE	3,9%
ARMAZENAGEM	0,7%
ADMINISTRAÇÃO	0,5%

Fonte – (??)

O quadro 1 representa como os custos em logística estão intrinsecamente ligados aos custos de transporte e um dos fatores que barram o desenvolvimento da logística no Brasil é justamente um conjunto de deficiências encontradas na infra-estrutura de transportes (??). Segundo dados do Instituto Brasileiro de Geografia e Estatística (IBGE) (??), a participação

do setor de transportes no PIB brasileiro foi de 4,4%, resultado de um volume de produção de R\$ 15,8 bilhões. No Brasil, o setor rodoviário é o mais expressivo em transporte de bens ou em oferecimento de serviços de logística, pois atinge praticamente todos os pontos do território nacional (??) e o Brasil tem uma das redes rodoviárias mais extensas do mundo, como mostrado na Figura 1. Segundo os levantamentos do Grupo de Estudos em Empresa Brasileira de Planejamento de Transportes (GEIPOT) (??), em 1999 as rodovias brasileiras tiveram um comprimento total de 1,72 milhões de quilômetros rodando nelas 1,84 milhões de veículos de carga, movimentando 447 bilhões de tonkm de cargas.

Figura 1 – As 10 redes rodoviárias mais extensas do Mundo

Rank	Países	Rodovias	Informação
		(km)	
1	Mundo	68,937,575	2008
2	Estados Unidos da América	6.465.799	2007
3	União Europeia	5.454.446	2008
4	Índia	3.316.452	2006
5	China	1.930.544	2005
6	Brasil	1.751.868	2004
7	Japão	1.196.999	2006
8	Canadá	1.042.300	2006
9	França	951.500	2006
10	Rússia	933.000	2006
11	Austrália	812.972	2004

Fonte – Central Intelligence Agencia – CIA - USA, 2008

A malha ferroviária brasileira é bem menos expressiva que a malha rodoviária. O Brasil possui a décima primeira maior malha ferroviária do mundo, com um total de 29,817 km. O Brasil está atrás de países de muito menor extensão geográfica, como Argentina, África do Sul e França.

Uma pesquisa mais recente, realizada em 2014 pela Fundação Dom Cabral (??) sobre os Custos Logísticos no Brasil, cujo objetivo foi avaliar os custos logísticos para as empresas e seu impacto nos negócios. A pesquisa consultou 111 empresas brasileiras, cujo faturamento equivale a 17% do PIB. De acordo com o estudo, os custos logísticos no Brasil consomem 11,19% da receita das empresas, que revelam ter um alto nível de dependência de rodovias (85,6%), máquinas e equipamentos (68,5%) e energia elétrica (66,7%). Ainda segundo a mesma pesquisa, o transporte continua sendo o fator muito presente na composição do custo logístico, basta ver, por exemplo, que 48,6% das empresas consultadas consideram muito alta a influência dos transportes na formação do preço final de seus produtos. Alguns dados da pesquisa

reforçam este aspecto. O transporte de longa distância é, com 44%, o fator mais representativo na estrutura de custo logístico das companhias, seguido de armazenagem, com 19,06%. Os transportes de produto acabado e de matéria-prima têm os maiores custos logísticos. Para 69,3% das empresas, o transporte de produto acabado tem um custo muito alto ou alto; e 61,2% das empresas têm a mesma opinião sobre o transporte de matéria-prima.

Neste contexto, as rodovias têm um papel de destaque. No transporte de insumos e produtos, o modal rodoviário é, de longe, o mais utilizado (81,7%), seguido do ferroviário (14,6%). Tanto que, para 40% das empresas consultadas, a melhoria das condições das rodovias é um fator importante para reduzir os custos logísticos. Nos modais ferroviário e portuário, a situação é ainda pior: 95,5% e 80,8% das companhias responderam que a qualidade nesses modais é muito ruim ou ruim, respectivamente.

Um exemplo prático de um problema de logística pode ser o exemplo do problema de coleta de lixo em uma cidade. Segundo o IBGE no ano 2000, o Brasil possuía 5507 municípios e 5471 municípios dispunham deste serviço de coleta de lixo. Diariamente foram coletados 228.413 toneladas de lixo nesse período (??). Estes dados mostram a dimensão deste problema, já que, sua relevância se torna ainda maior quando este problema é aplicado a grandes centros urbanos com diversos centros de coleta de lixo, centros de tratamento do lixo e transporte para destino final. Estas marcas surpreendentes colocam o Brasil entre os maiores produtores de lixo do mundo, com dispêndio elevadíssimo da ordem de R\$ 4 bilhões por ano. O custo da coleta somente com equipamentos e pessoal indica ser aproximadamente 50% deste total (??).

A grande quantidade de recursos alocados para as operações de logísticas em diversas áreas da sociedade têm interessado pesquisadores no sentido de estudar formas de se otimizar este processo. Modelos de otimização têm sido criados para resolver diversos problemas de logística da atualidade. Entre os problemas de logística existentes, chama-se atenção para os problemas de roteamento de arcos, que são problemas que estão na classe de Problemas do Carteiro Chinês e consistem em cobrir todos os arcos de um grafo, minimizando a distância total percorrida. Estes tipos de problema surgem em diversos contextos práticos onde se deseja otimizar rotas. Alguns exemplos de aplicações que envolvem roteamento de arcos são os seguintes (??):

- Varrição e lavagem de ruas (??);
- Coleta de lixo doméstico;
- Fiscalização de linhas de ônibus;
- Serviços de transporte escolar;

- Serviço de entrega de cartas e encomendas de correios;
- Testes topológicos de sistemas de computadores (??);
- Minimização de número de vias, no desenho de circuitos VLSI (??);
- Inspeção de estruturas metálicas utilizando robôs (??).

Como a maioria dos gastos de logística do Brasil são gastos com transporte e como a logística de transporte é baseada essencialmente em transporte rodoviário é essencial para um maior lucro de produtores e transportadores otimizar as rotas percorridas elaborando um modelo operacional que satisfaça as necessidades (??), ou seja, é importante cobrir os arcos da melhor maneira possível, o que corresponde a resolver o Problema do Carteiro Chinês.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Este trabalho tem como objetivo geral estudar algumas variações do Problema do Carteiro Chinês, e contribuir às suas soluções com um algoritmo exato capaz de calcular qualquer tipo de PCC e capaz de mostrar o percurso do carteiro de uma maneira gráfica.

2.2 OBJETIVOS ESPECÍFICOS

- a) Explorar o Problema do Carteiro Chinês, abordando todos os tipos de PCC existentes;
- b) Implementar os conceitos do Problema do Carteiro Chinês para calcular qualquer tipo de PCC;
- c) Realizar testes com o PCC implementado;
- d) Comparar os resultados obtidos com trabalhos apresentados na literatura.

2.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte maneira:

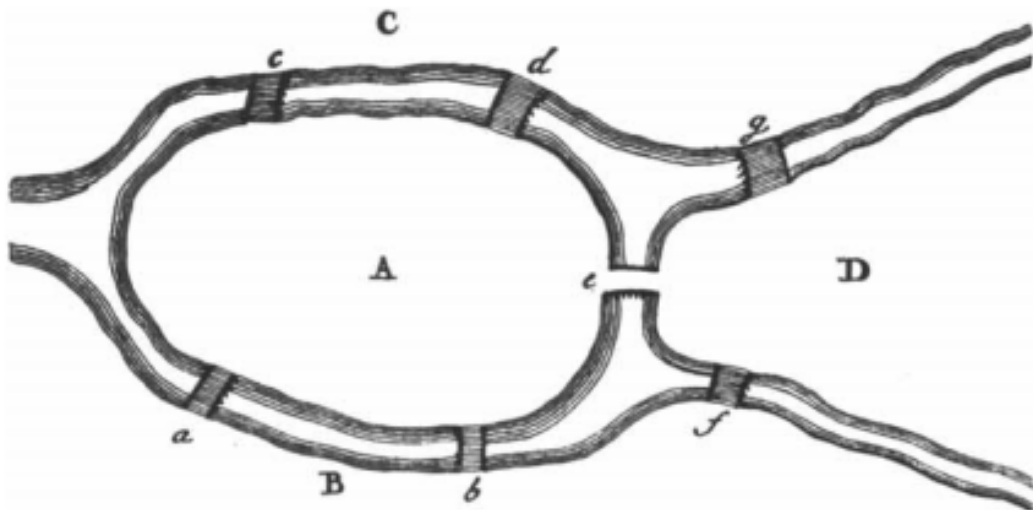
- No capítulo 3 é realizado um estudo geral sobre o Problema do Carteiro Chinês mostrando o que há na literatura sobre todos os tipos de PCC existentes;
- No capítulo 5 são citados e comentados os trabalhos relacionados;
- No capítulo 4 é explicado a metodologia utilizada para criar o algoritmo proposto que calcula o PCC;
- No capítulo 6 são realizados testes com o algoritmo criado, comparando-o com outras soluções já desenvolvidas;
- O capítulo 7 é a conclusão do trabalho e são sugeridos trabalhos futuros que poderão ser realizados a partir deste trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 HISTÓRICO

Os problemas de percurso em arcos de um grafo estão entre os problemas mais antigos da teoria dos grafos. Um dos primeiros problemas de percurso em grafos estudado foi o problema das pontes de Königsberg (??). Uma representação deste problema é mostrado na Figura 2. Este problema consistia em checar a existência de um caminho fechado que atravessasse exatamente uma única vez cada uma das sete pontes da cidade de Königsberg.

Figura 2 – As sete pontes de Königsberg

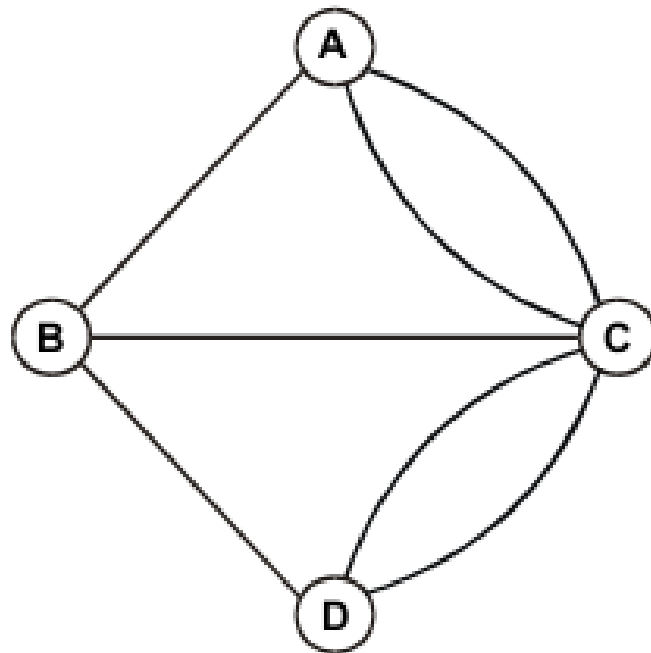


Fonte – Elaborado pelo autor.

O matemático suíço Leonhard Euler estudou este problema e enumerou as condições necessárias para que existisse um caminho fechado em um grafo (??). Os grafos que possuem este caminho fechado são grafos que têm uma propriedade interessante, todos os seus vértices têm grau par. Tais grafos que possuem esta característica foram denominados de grafos eulerianos, em homenagem a Euler. Sandifer alega que este foi o estudo mais famoso de Euler (??), apesar dos seus 25 livros e 850 artigos técnicos publicados (??). Este estudo foi de tamanha importância que fundamentou o início da teoria dos grafos. A Figura 3 mostra um grafo que representa as pontes da cidade de Königsberg.

Muitos anos depois de Euler ter estudado este problema, um matemático chamado Kwan Mei-Ko, que trabalhava nos correios da China se atentou em resolver o mesmo problema antes resolvido por Euler porém adaptado para a realidade dos correios (??). Kwan adaptou o problema estudado por Euler considerando que as pontes a serem atravessadas agora seriam as

Figura 3 – Grafo representativo das pontes de Königsberg



Fonte – Elaborado pelo autor.

ruas percorridas pelos carteiros, otimizando assim o seu percurso. Além de definir a travessia do carteiro, Kwan trabalhou em como realizar esta travessia da maneira mais econômica possível (??). Ao contrário do problema de Euler, que se preocupa com a existência de um circuito euleriano, que passa exatamente uma vez por cada aresta, a questão abordada por Kwan era em relação aos grafos não-eulerianos, na tentativa de identificar o circuito de distância mínima que passa pelo menos uma vez por cada aresta, num grafo qualquer (??), ou seja, ele tentou encarar o problema sob uma ótica de otimização, buscando saber como realizar o circuito do carteiro com o menor custo possível. Kwan não conseguiu achar uma solução satisfatória para o problema, porém seu trabalho se mostrou tão revelante na comunidade científica que foi denominado como Problema do Carteiro Chinês (PCC).

Estudar detalhadamente o Problema do Carteiro Chinês exige que tenhamos conhecimento de Teoria dos Grafos. É importante conhecermos a terminologia e alguns resultados importantes referentes a grafos, sendo um tema rico em aplicações. Por este motivo a próxima seção será destinada a tratar destes conceitos.

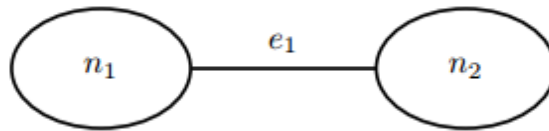
3.2 DEFINIÇÕES

A maioria das definições e termos relacionados à Teoria de Grafos usados nesse trabalho é padrão e pode ser encontrada nos trabalhos clássicos, como o de Christofides (??).

Um **grafo** é representado por uma Tupla $G = (N, A, E)$, onde $N = \{n_1, n_2, \dots, n_n\}$ é o conjunto de **vértices** (ou **nós**) do grafo, $A = \{a_1, a_2, \dots, a_n\}$ é o conjunto de **arcos** do grafo e $E = \{e_1, e_2, \dots, e_n\}$ é o seu conjunto de **arestas**. A padronização da cardinalidade destes conjuntos neste trabalho será a seguinte, $n = |N|$, $r = |A|$ e $m = |E|$. Outra definição recorrente diz respeito ao conjunto de **links** de G . Este conjunto é definido por $L = A \cup E$ e $L = \{l_1, l_2, \dots, l_n\}$, ou seja, um link $l \in L$ pode ser um arco ou uma aresta do grafo G .

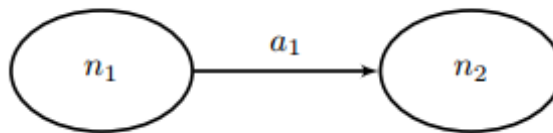
Um link pode ser descrito como uma par de vértices (n_i, n_j) que indica que o vértice n_i e o vértice n_j são os vértices **terminais** do link. No caso de o link ser uma aresta, não importa a ordem de vértices terminais, como mostrado na Figura 4. Já no caso de o link ser um arco, a ordem se dá do vértice **inicial** para o vértice **final**, como mostrado na Figura 5.

Figura 4 – Aresta e_1 que liga os vértices n_1 e n_2



Fonte – Elaborado pelo autor.

Figura 5 – Arco a_1 que liga os vértices n_1 e n_2



Fonte – Elaborado pelo autor.

A Figura 6 mostra um grafo com dois links l_1 e l_2 , sendo que l_1 é uma aresta do grafo, ou seja, $l_1 \in E$, e l_2 é um arco do mesmo grafo, portanto $l_2 \in A$.

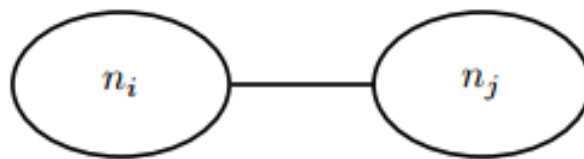
Em algumas situações é interessante observar uma determinada aresta (n_i, n_j) como se esta fosse um par de arcos contrariamente direcionados (n_i, n_j) e (n_j, n_i) , como mostrado nas Figuras 7 e 8.

Figura 6 – Links l_1 e l_2 que ligam os vértices n_1 , n_2 e n_3



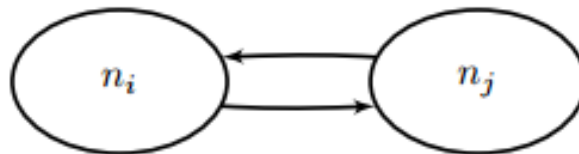
Fonte – Elaborado pelo autor.

Figura 7 – Aresta (n_i, n_j)



Fonte – Elaborado pelo autor.

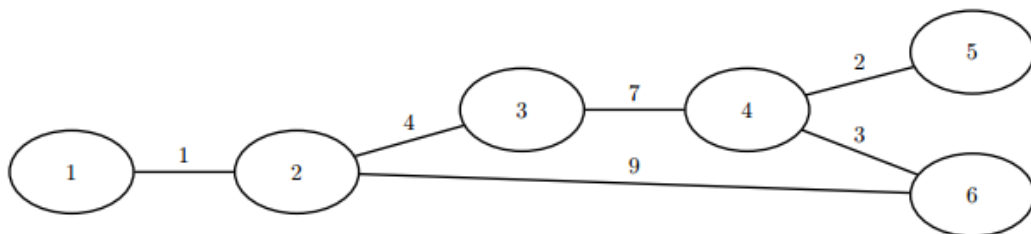
Figura 8 – Arcos (n_i, n_j) e (n_j, n_i) contrariamente direcionados



Fonte – Elaborado pelo autor.

A Figura 9 mostra um **grafo valorado** (ou **ponderado**). Neste tipo de grafo os links possuem um custo para ser percorrido.

Figura 9 – Exemplo de grafo valorado



Fonte – Elaborado pelo autor.

Outro conceito importante de destacar é o conceito de vértices **adjacentes**. Dois vértices n_i e n_j serão adjacentes se existir o link (n_i, n_j) . A **vizinhança** de um vértice n_i é o conjunto de vértices adjacentes a n_i .

A cada link de um grafo pode ser associado um **custo** d_{ij} para atravessar este link. A

matriz $D = [d_{ij}]$ é a **matriz de custos** do grafo, onde d_{ij} é o custo do link $(n_i, n_j) \in L$ e $d_{ij} = \infty$ se $(n_i, n_j) \notin L$, ou seja, se não existir o link entre n_i e n_j seu custo associado será infinito, pois é impossível atravessar um link que não existe.

A partir do que foi mostrado até este momento podemos realizar as seguintes formulações:

- Se $E = \emptyset$ então G é um grafo **dirigido** (orientado), pois só possui arcos e não possui arestas;
- Se $A = \emptyset$ então G é um grafo **não dirigido** (não orientado), pois só possui arestas e não possui arcos;
- Se $E \neq \emptyset$ e $A \neq \emptyset$ então G é um grafo **misto**, pois possui tanto arestas como arcos.

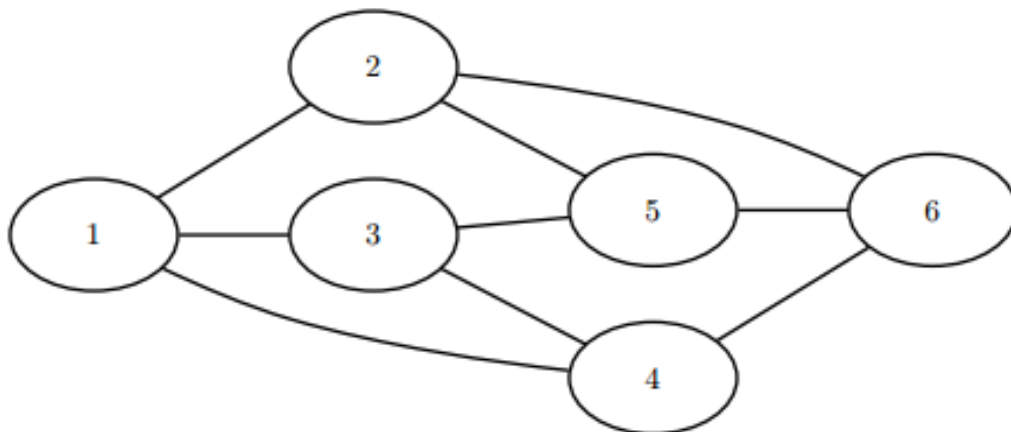
Um grafo G é **completo** se para qualquer vértice $n_i \in N$ e $n_j \in N$ existir o link (n_i, n_j) , ou seja, existe um link (aresta ou arco) entre todos os vértices do grafo.

Um **subgrafo** G_s de G é o grafo (N', L') com $N' \subset N$ e $L' = \{(n_i, n_j) | (n_i, n_j) \in L, n_i \in N', n_j \in N'\}$. Portanto, um subgrafo de G é um grafo contendo um subconjunto de vértices de G , porém com todos os links que conectam estes vértices no grafo original.

Um **grafo parcial** G_p de G é o grafo (N, L') com $L' \subset L$. Portanto, um grafo parcial é um grafo com os mesmos vértices do grafo original, porém com apenas um subconjunto próprio de links do grafo original. Um grafo parcial também é chamado de **subgrafo gerador**.

A Figura 9 mostra um subgrafo gerador do grafo apresentado na Figura 10.

Figura 10 – Subgrafo gerador do grafo da Figura 10



Fonte – Elaborado pelo autor.

Em um grafo não dirigido, para cada vértice n_i define-se o **grau** $d(n_i)$ como o número de arestas que incidem no vértice n_i . Quando o grafo for dirigido, existirão dois tipos distintos

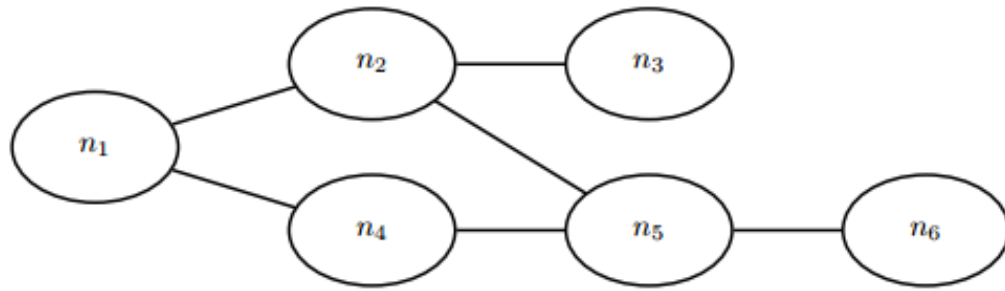
de grau: **grau de entrada** e **grau de saída**. O grau de entrada de um vértice $d_e(n_i)$ é definido como o número de arcos cujo vértice final é o vértice n_i . Já o grau de saída de um vértice $d_s(n_i)$ é o número de arcos cujo vértice inicial é o vértice n_i . Se um vértice tiver o grau de entrada igual ao grau de saída ele é dito **balanceado**.

Um **caminho** em um grafo não dirigido é uma sequência de arestas que conectam vértices distintos do grafo. Já em um grafo dirigido, um **caminho dirigido** (**caminho orientado**) é uma sequência de arcos que conectam vértices do grafo adicionando a restrição que os arcos devem possuir a mesma direção. Desta forma um caminho $C(n_i, n_j)$ se inicia no vértice n_i e termina no vértice n_j , onde n_i e n_j não são necessariamente adjacentes. Um **passeio** é uma sequência de arestas (ou arcos) que conectam vértices porém é possível passar por um mesmo vértice mais de uma vez. Tanto no caminho quanto no passeio o vértice inicial e o final são chamados respectivamente de **vértice origem** e **vértice destino** e os outros vértices são chamados de **vértices intermediários**. Um caminho é **fechado** se o vértice inicial for igual ao vértice final, além disso, um caminho fechado também é chamado de **circuito**. Uma **cadeia** é uma sequência de links, também ligando dois nós não necessariamente adjacentes, sem respeito à orientação dos arcos.

Um vértice n_i é acessível a partir de n_j se existir um caminho entre n_i e n_j . Um grafo é **fortemente conexo** (**f-conexo**) (??), se para qualquer par de vértices n_i e n_j existe pelo menos um caminho que conecta n_i e n_j , ou ainda, um grafo fortemente conexo é aquele em que todo vértice deve ser alcançável de qualquer outro vértice do grafo (??). Em um grafo fortemente conexo, dois vértices quaisquer são mutuamente acessíveis. Um grafo é dito **conexo**, ou **fracamente conexo**, se para qualquer par de nós n_i e n_j existe pelo menos uma cadeia que conecta n_i e n_j . Todo grafo não dirigido e conexo é f-conexo. Se pelo menos para um par de vértices tal cadeia não existe, o grafo então é **desconexo**. As Figuras 11, 12 e 13 mostram respectivamente um grafo não dirigido e conexo, outro grafo não dirigido porém desconexo e um grafo dirigido e fortemente conexo.

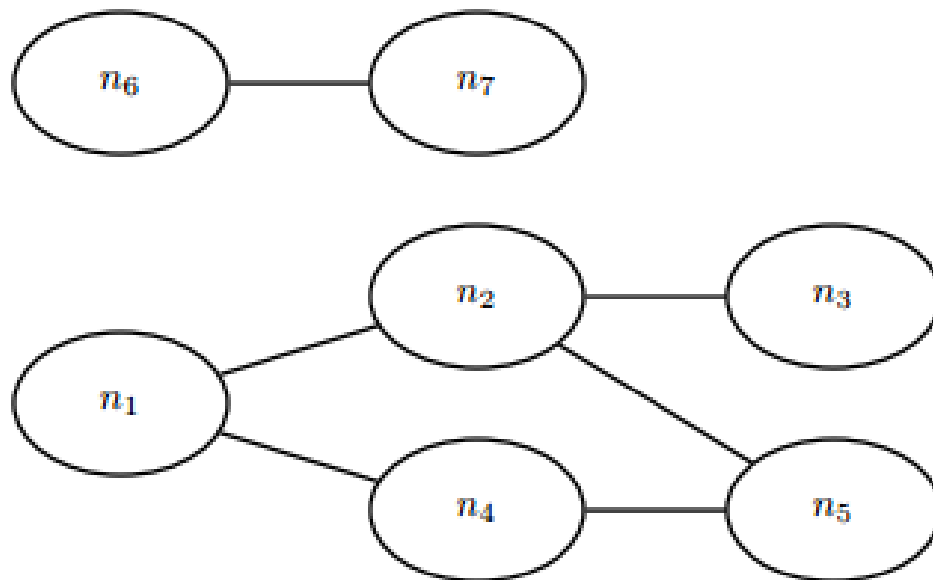
Um **circuito euleriano** em um grafo é um circuito que passa por todos os links de um grafo, sem que repita o mesmo link mais de uma vez. Um grafo G é dito **Euleriano** (ou **Unicursal**) se ele possui pelo menos um circuito euleriano. Um **caminho euleriano** é um caminho que passa exatamente uma vez por cada aresta do grafo. Um grafo que não contém um circuito euleriano, mas contém um caminho euleriano é denominado **semi-euleriano**. As Figuras 14 e 15 são exemplos de um grafo euleriano e de um semi-euleriano, respectivamente.

Figura 11 – Grafo não dirigido e f-conexo



Fonte – Elaborado pelo autor.

Figura 12 – Grafo não dirigido e desconexo



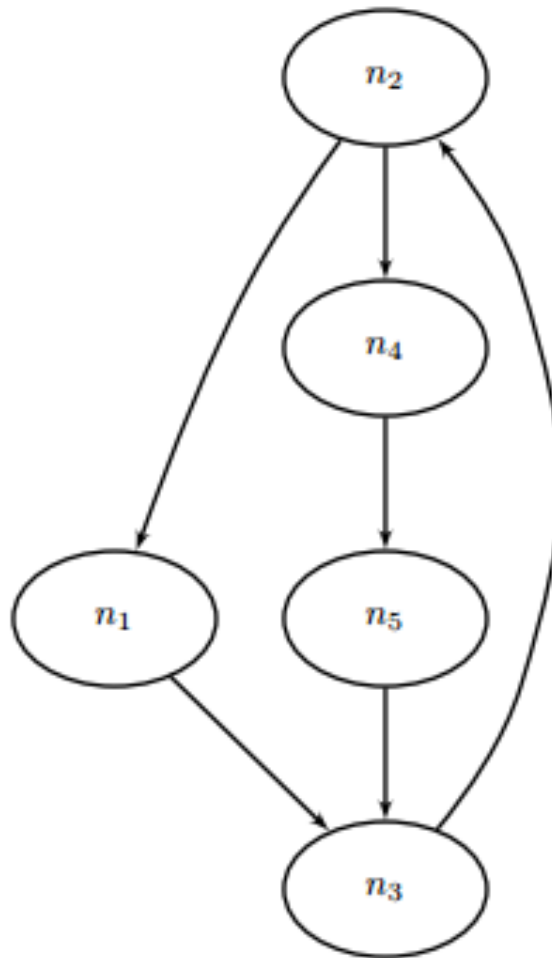
Fonte – Elaborado pelo autor.

3.3 CIRCUITOS EULERIANOS

Como mencionado acima um circuito euleriano em um grafo é um circuito que passa por todos os links de um grafo, sem que repita o mesmo link mais de uma vez. Para que um grafo G tenha um circuito euleriano ele deve (????):

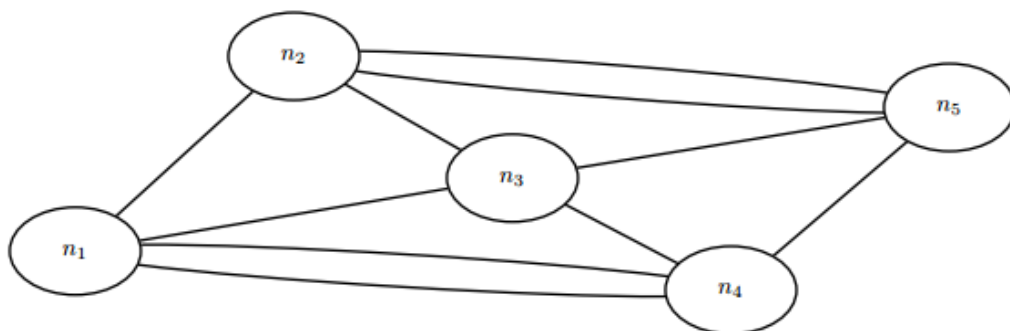
- Se G for não dirigido:
 - G deve ser fortemente conexo e todos os vértices devem possuir grau par
- Se G for dirigido:
 - G deve ser fortemente conexo e todos os vértices devem ter grau de entrada igual ao grau de saída

Figura 13 – Grafo dirigido e fortemente conexo



Fonte – Elaborado pelo autor.

Figura 14 – Grafo euleriano



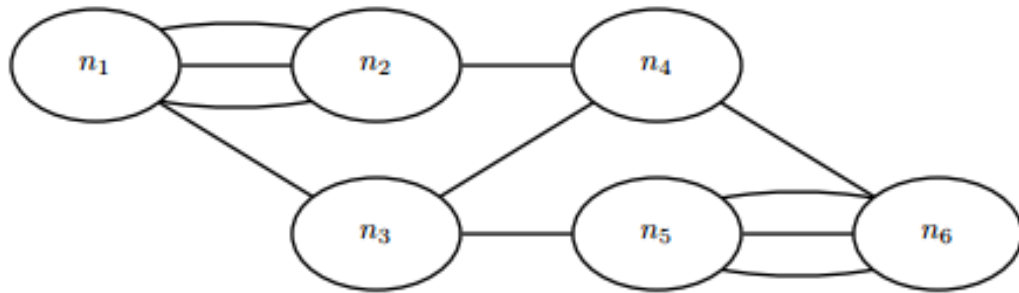
Fonte – Elaborado pelo autor.

- Se G for misto:

- G deve ser fortemente conexo, todos os vértices devem ter grau par e devem ser balanceados

Para se obter um caminho euleriano as condições são praticamente as mesmas de um circuito euleriano, porém pode-se obter um caminho euleriano se o número de vértices com

Figura 15 – Grafo semi-euleriano



Fonte – Elaborado pelo autor.

grau ímpar (ou vértices desbalanceados) for igual a dois. Nestes casos específicos, o caminho se iniciará em um dos vértices ímpares (ou desbalanceados) e terminará no outro.

O teorema básico sobre a existência de um circuito euleriano, em um grafo não dirigido, é o seguinte:

Teorema 3.3.1 (Euler) *Um grafo fortemente conexo $G(N, E)$ contém um circuito euleriano, se, e somente se, todos os graus dos vértices são pares.*

A prova do teorema é descrita a seguir. Se G contém um circuito euleriano, se contarmos para cada vértice, a entrada e saída dele, ao final de todo percurso teremos um conjunto de números pares, tendo em vista que qualquer circuito euleriano deve usar uma aresta para entrar em cada vértice, e uma outra para partir do mesmo. Suponha agora que temos um grafo G onde todos os seus vértices têm grau par. Escolha um vértice n_i qualquer e comece a percorrê-lo sem repetir arestas, adicionando as arestas percorridas no conjunto C , até não existirem arestas a serem percorridas, a partir do vértice corrente. Como todos os vértices têm grau par então o último vértice alcançado será justamente o n_i . Se o circuito C contiver todas as arestas de G então a demonstração está concluída, caso contrário existirão arestas não percorridas. Se ainda restarem arestas não percorridas, existirá algum caminho entre algum vértice do circuito até uma aresta q não incluída em C , já que o grafo é conexo. Imagine que este caminho seja formado pela aresta (n_j, n_k) , onde n_j pertence ao circuito C e n_k pertence a aresta q . Se isto ocorrer deve-se percorrer o grafo a partir de n_j visitando todas as novas arestas sem acessar nenhuma aresta em C . Este novo circuito C' pode ser unido ao circuito C formando um único circuito. Agora basta percorrer C a partir de n_j e quando retornar a n_j começar a percorrer C' . Repete-se este processo até que todas as arestas tenham sido visitadas. No final teremos um único

circuito formado pela união de vários circuitos. O circuito resultante é chamado euleriano, assim como o grafo. Esta prova é realizada de maneira análoga para o caso de o grafo ser dirigido.

Um teorema interessante dos grafos não eulerianos é o seguinte:

Teorema 3.3.2 ((??)) *Todo grafo tem uma quantidade par de vértices com grau ímpar.*

Este fato pode ser explicado da seguinte maneira, considerando m o número de arestas do grafo, sabemos que cada aresta liga dois vértices do grafo, então a soma dos graus de todos os vértices é $2m$ (??). Portanto, se $S_p + S_i = 2m$, onde S_p é a soma dos graus dos vértices pares e S_i é a soma dos graus dos vértices ímpares e como S_p é obviamente um número par, S_i também tem que ser um número par, e se a soma de alguns números ímpares é par, é porque a quantidade destes números ímpares é par, portanto a quantidade de vértices ímpares é par.

3.3.1 Algoritmos para encontrar Circuitos Eulerianos

3.3.1.1 Algoritmo de Hierholzer

A prova do teorema de Euler nos fornece uma boa maneira de, dado um grafo euleriano, encontrar nele um circuito euleriano. Esta forma descrita acima é formalizado em um algoritmo descrito por Hierholzer (??) e mostrado no Algoritmo 3.

A ideia é, a partir de um vértice qualquer, percorrer arestas até retornar ao vértice inicial. Porém, Desta forma, pode ser obtido um ciclo que não inclua todas as arestas do grafo. Enquanto houver um vértice que possui arestas ainda não exploradas, comece um caminho neste vértice e tente voltar a ele, usando somente arestas ainda não percorridas.

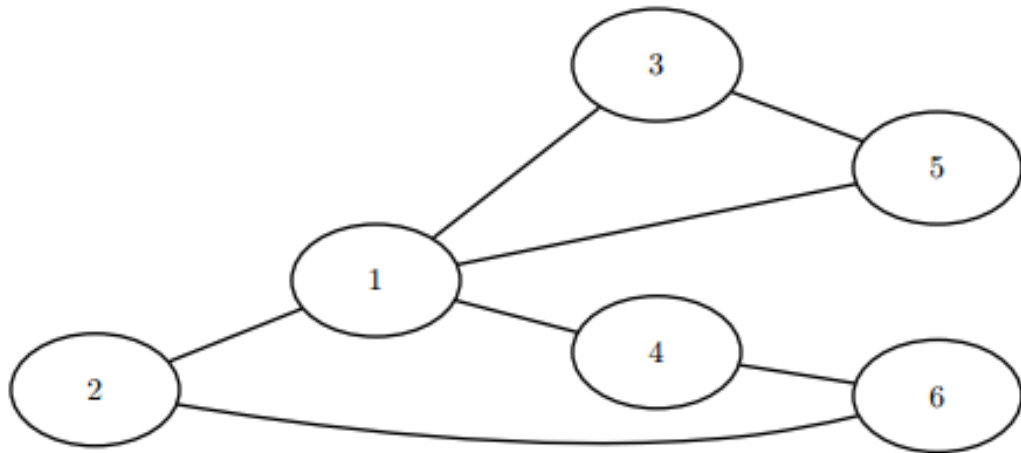
As Figuras 16, 17, 18 e 19 mostram um exemplo do funcionamento do algoritmo de Hierholzer.

A Figura 16 mostra o grafo utilizado para o exemplo. Para este exemplo foi escolhido o vértice 1 como vértice inicial.

A Figura 17 mostra a primeira iteração do algoritmo, onde, iniciando do vértice 1, percorre a aresta 1-5, 5-3 e retorna ao vértice 1 percorrendo a aresta 3-1, terminando assim a primeira iteração com o conjunto $C' = [1, 5, 3, 1]$ e, como se trata da primeira iteração, $C = C'$.

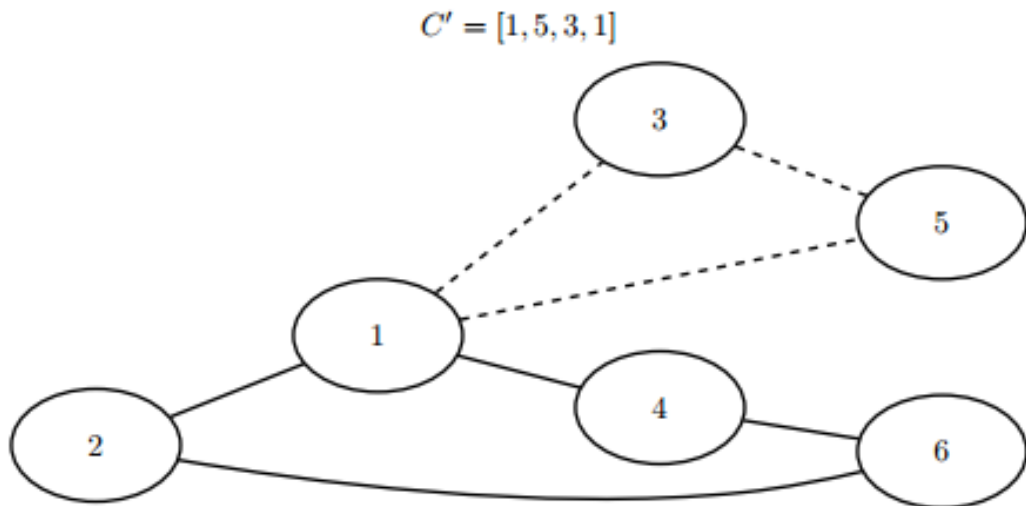
Como o vértice 1 ainda possui arestas não percorridas, a segunda iteração do algoritmo se inicia por este vértice. A Figura 17 mostra esta iteração que se finaliza com o conjunto $C' = [1, 2, 6, 4, 1]$. Esta iteração é mostrada na Figura 18.

Figura 16 – Grafo euleriano utilizado para descrever o algoritmo de Hierholzer



Fonte – Elaborado pelo autor.

Figura 17 – Primeira iteração do algoritmo de Hierholzer



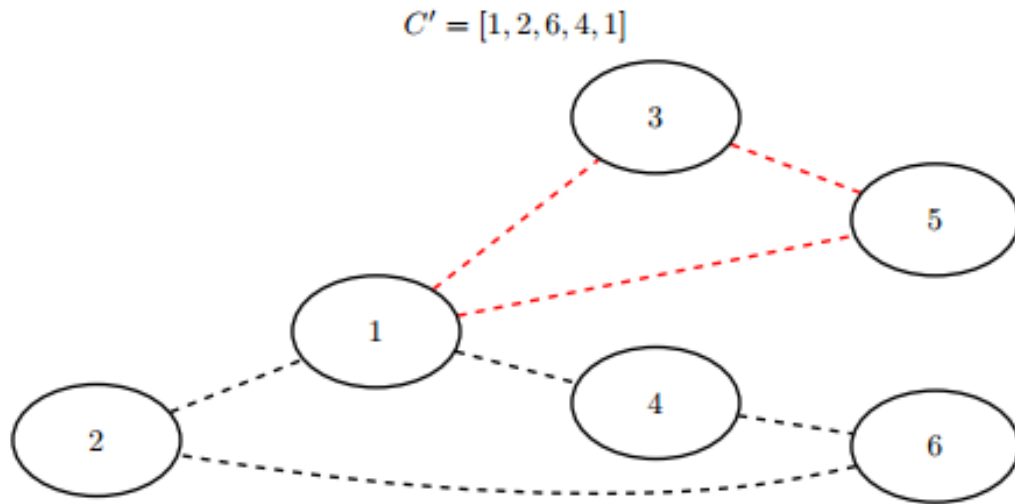
Fonte – Elaborado pelo autor.

O conjunto C é atualizado com o C' da última iteração terminando esta iteração com $C = [1, 5, 3, 1, 2, 6, 4, 1]$. Como todas as arestas foram visitadas, o algoritmo termina e C contém o circuito euleriano desejado, como mostrado na Figura 19.

3.3.1.2 Algoritmo de Fleury

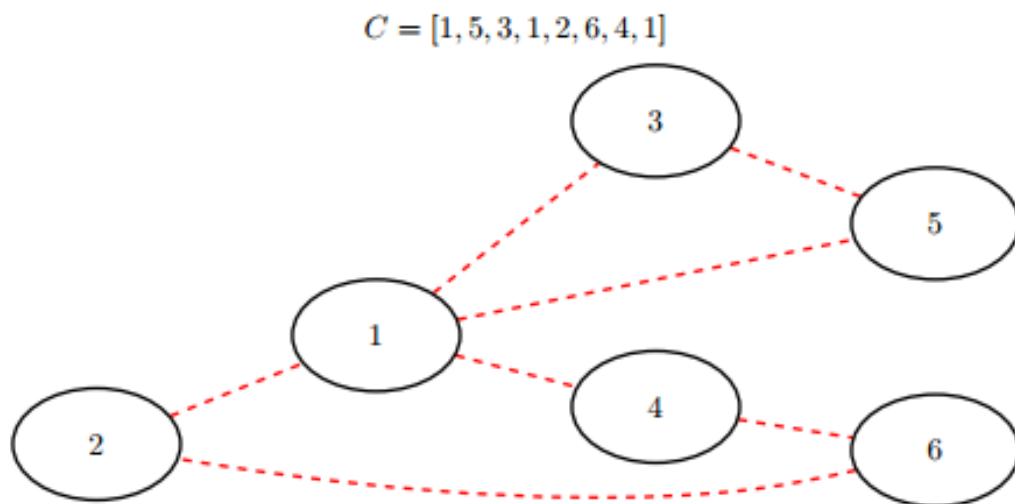
Uma forma mais simples de encontrar um circuito euleriano de um grafo euleriano é aplicar o algoritmo de Fleury (??), descrito no Algoritmo 4. A ideia deste algoritmo é partindo de um vértice inicial escolher uma aresta dele e percorrê-la se ela não for uma **ponte**, ao percorrer esta aresta, escolher nova aresta que não seja ponte a partir do novo vértice de partida e após atravessar cada aresta atualizar o circuito do carteiro com o vértice visitado. A única vez que

Figura 18 – Segunda iteração do algoritmo de Hierholzer



Fonte – Elaborado pelo autor.

Figura 19 – Resultado do algoritmo de Hierholzer



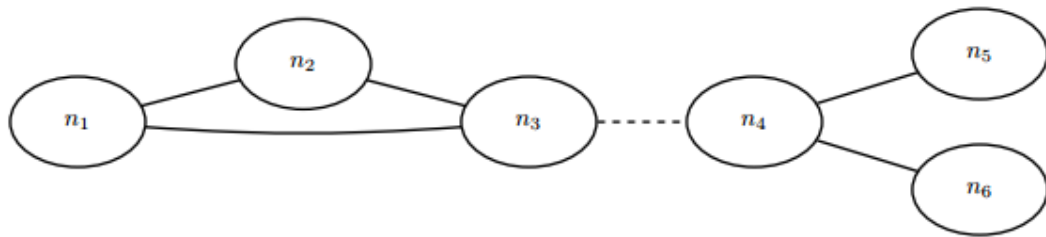
Fonte – Elaborado pelo autor.

uma aresta ponte será escolhida para ser atravessada é quando ela for a única opção possível. Uma ponte em um grafo é uma aresta que se retirada do grafo, torna o grafo desconexo. Na Figura 20, a retirada da aresta (n_3, n_4) torna o grafo desconexo, como mostrado na Figura 21.

Um exemplo do algoritmo de Fleury é mostrado a seguir. O grafo utilizado como exemplo é o mesmo do algoritmo de Hierholzer (Figura 16). As Figuras 22, 23, 24, 25, 26, 27 e 28 demonstram o grafo durante as iterações do algoritmo.

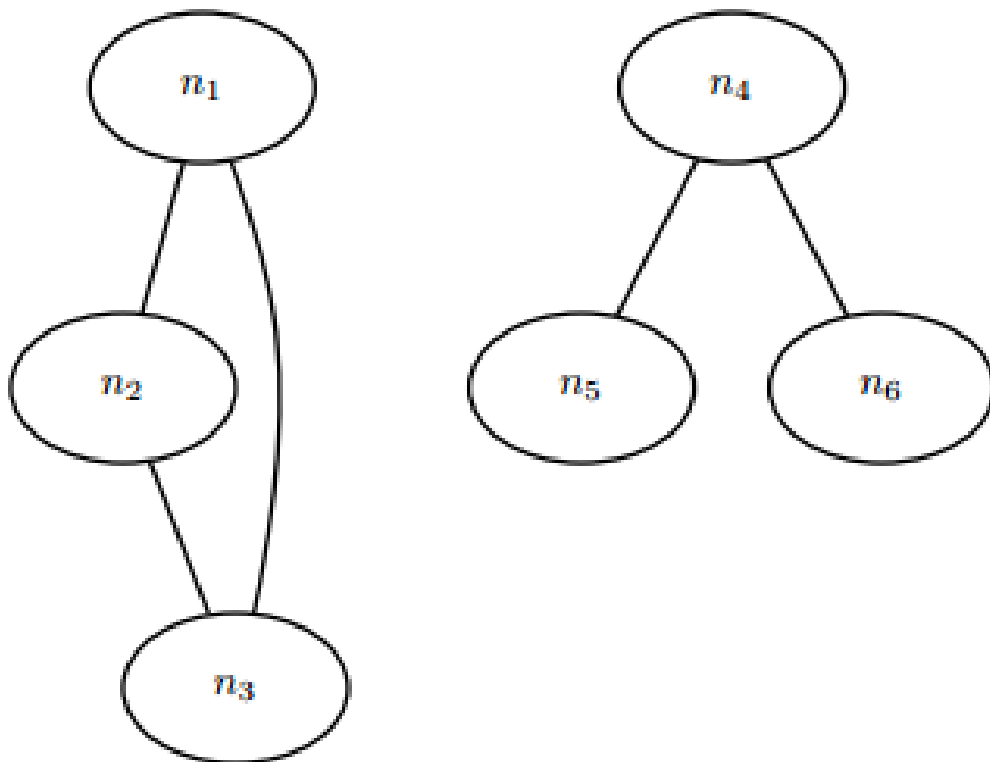
Edmonds (??) encontra um circuito euleriano de uma maneira diferente dos algoritmos descritos anteriormente, ele opta pela construção de uma arborescência para construir tal circuito euleriano.

Figura 20 – Grafo com uma aresta que é uma ponte



Fonte – Elaborado pelo autor.

Figura 21 – Grafo desconexo após retirada da aresta que era uma ponte



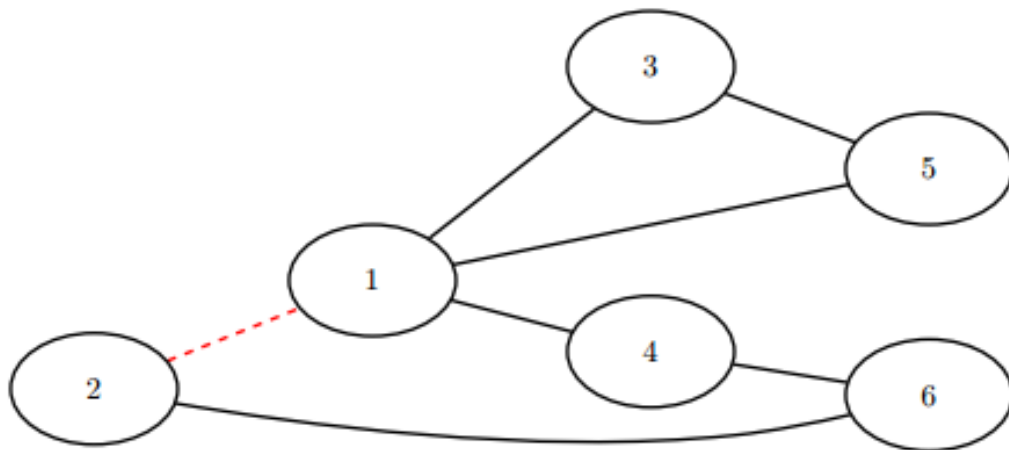
Fonte – Elaborado pelo autor.

3.4 PROBLEMA DO CARTEIRO CHINÊS

Dado um grafo $G = (N, L)$, conexo, onde a cada link $l \in L$ é associado um custo d_l , o **Problema do Carteiro Chinês (PCC)** é definido por Kwan (??) da seguinte maneira: "Um carteiro tem que cobrir seu local de trabalho, antes de retornar ao posto. O problema é encontrar a menor distância de percurso para o carteiro". Na Teoria dos Grafos, o problema consiste em encontrar um circuito de custo mínimo, contendo todas as arestas, pelo menos uma vez (circuito de carteiro).

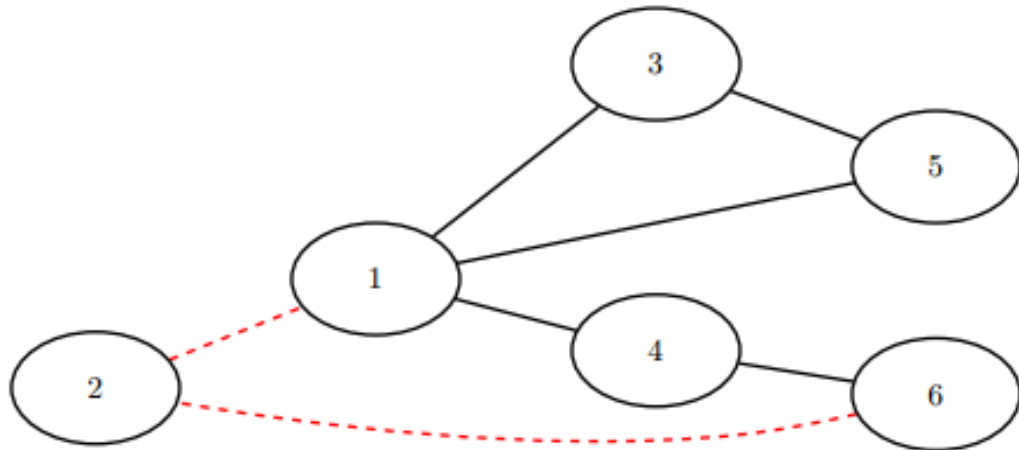
Para que o problema tenha solução é necessário que o grafo seja euleriano, portanto se o grafo já for euleriano, o problema se reduz a encontrar um circuito euleriano neste grafo, ou

Figura 22 – Primeira iteração do algoritmo de Fleury



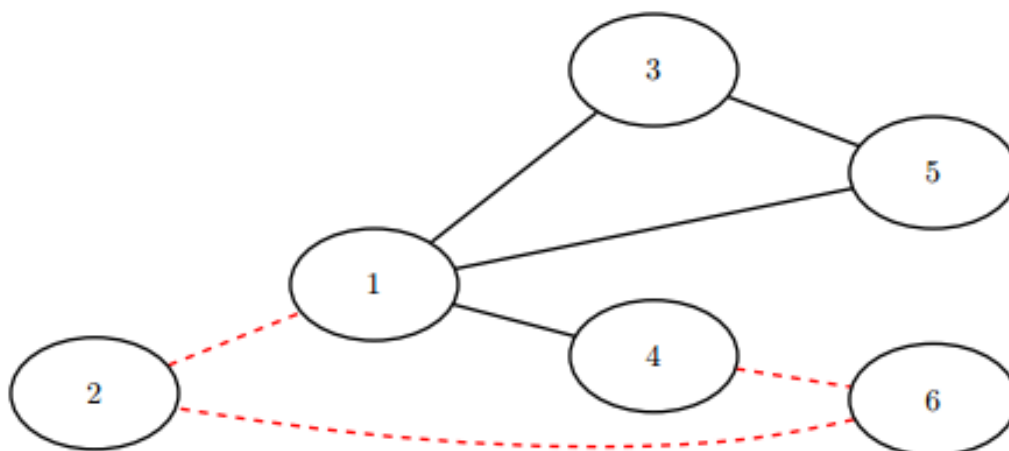
Fonte – Elaborado pelo autor.

Figura 23 – Primeira iteração do algoritmo de FleurySegunda iteração do algoritmo de Fleury



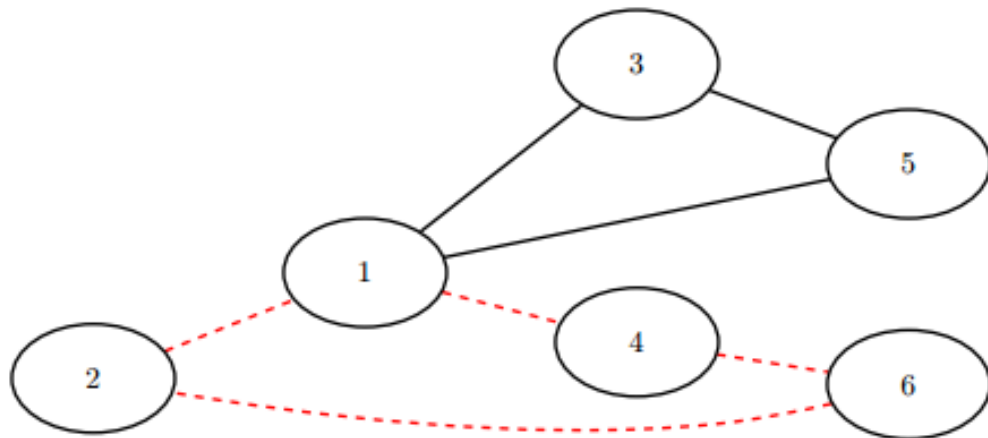
Fonte – Elaborado pelo autor.

Figura 24 – Terceira iteração do algoritmo de Fleury



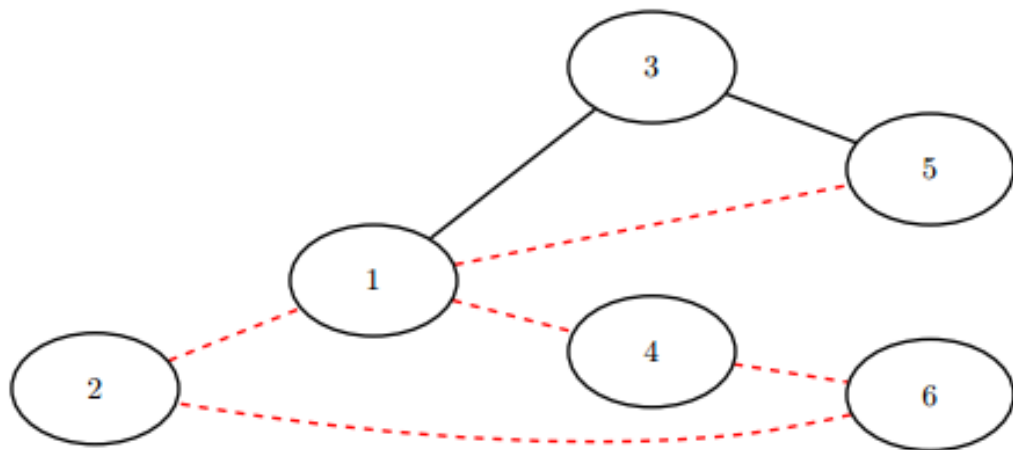
Fonte – Elaborado pelo autor.

Figura 25 – Quarta iteração do algoritmo de Fleury



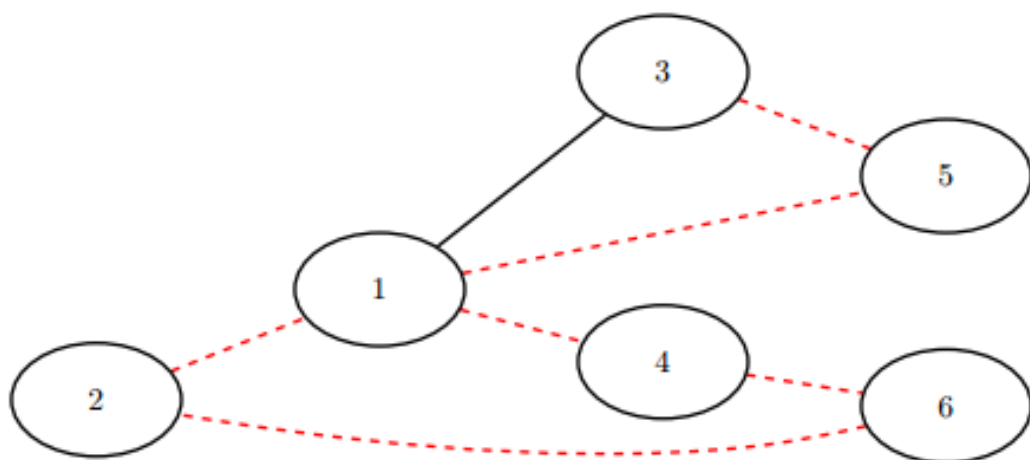
Fonte – Elaborado pelo autor.

Figura 26 – Quinta iteração do algoritmo de Fleury



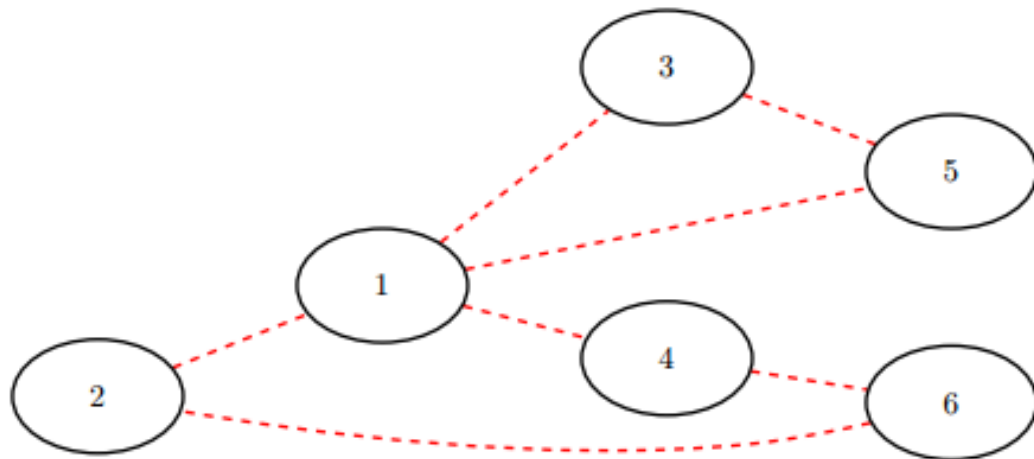
Fonte – Elaborado pelo autor.

Figura 27 – Sexta iteração do algoritmo de Fleury



Fonte – Elaborado pelo autor.

Figura 28 – Última iteração do algoritmo de Fleury



Fonte – Elaborado pelo autor.

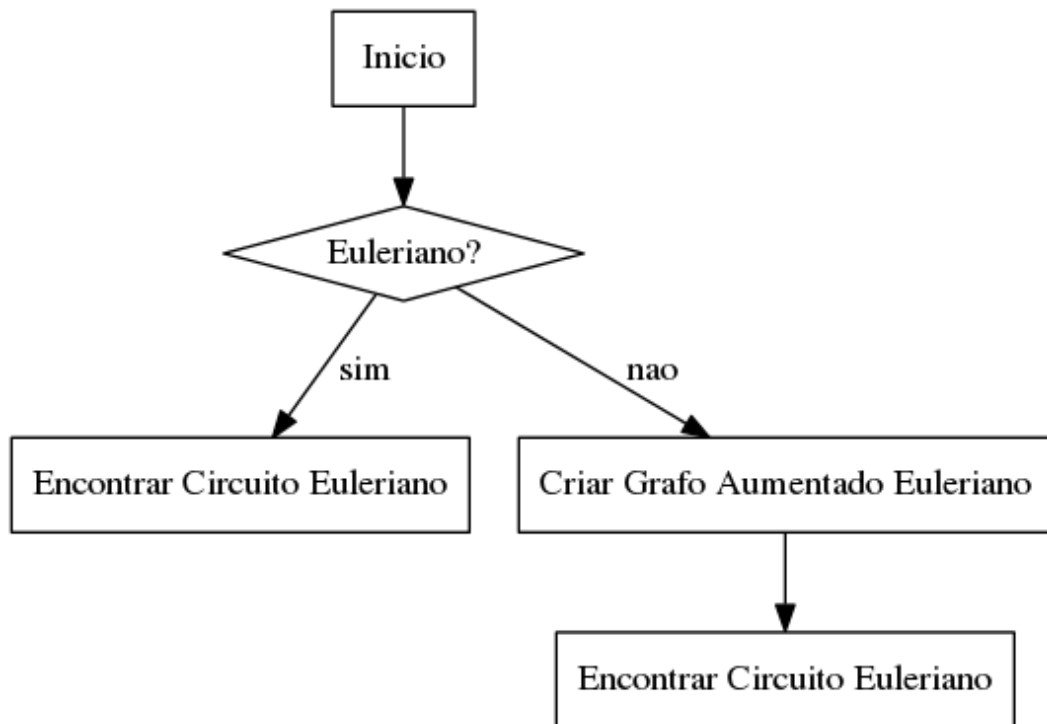
seja, algumas arestas ou arcos devem ser duplicados. Se o grafo não for euleriano será necessário adicionar cópias de arestas (ou arcos) no grafo para que este se torne euleriano. Neste caso o problema se torna descobrir quais arestas devem ser copiadas para que o circuito percorrida seja mínimo, e após adicionadas estas arestas e obtido um **grafo aumentado**, que é euleriano, basta encontrar o circuito euleriano dentro deste grafo.

O Fluxograma mostrado na Figura 29 exemplifica o fluxo de um algoritmo para encontrar um circuito euleriano em um grafo G dado.

A Figura 30 mostra um grafo originalmente não-euleriano, por conter dois vértices de grau ímpar. A duplicação das arestas num caminho que liga estes dois vértices torna o grafo euleriano. Numa solução ótima, entretanto, tais duplicações devem acontecer ao longo do caminho mais curto, o que não acontece neste exemplo.

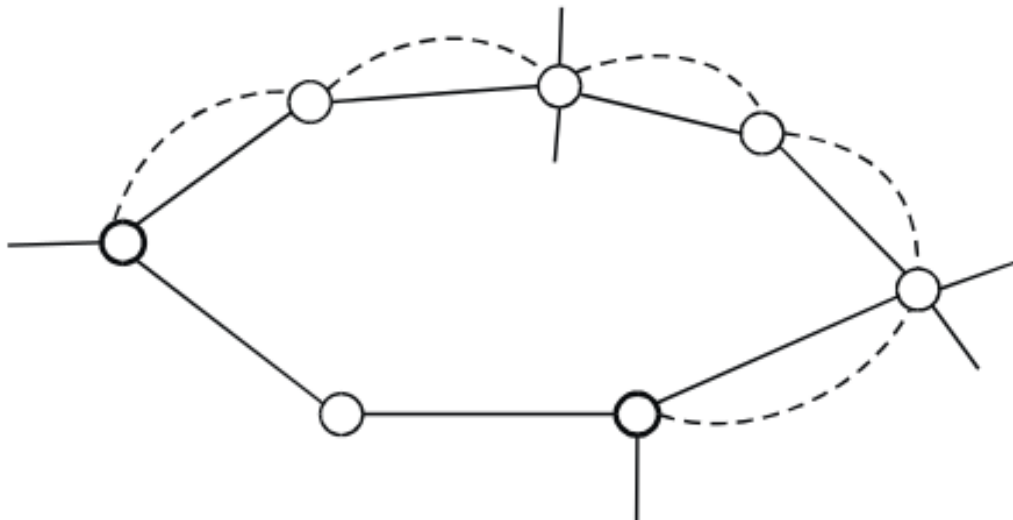
Quando o grafo tem mais que um par de vértices com grau ímpar, a duplicação de caminhos mínimos que ligam pares de vértices de grau ímpar, embora necessário, não é suficiente para garantir a otimalidade da solução. A Figura 31 mostra uma situação em que dois pares de vértices de grau ímpar são associados. Porém como não existem arestas entre estes pares vértices, são duplicadas as arestas que estão no caminho mínimo que liga estes vértices, como mostra a Figura 32. Entretanto, nesta figura os pares de vértices não estão associados de maneira ótima. A maneira ótima de associar estes vértices é mostrado na Figura 33.

Figura 29 – Fluxograma para encontrar um circuito euleriano em um grafo



Fonte – Elaborado pelo autor.

Figura 30 – Exemplo de Grafo Euleriano não ótimo

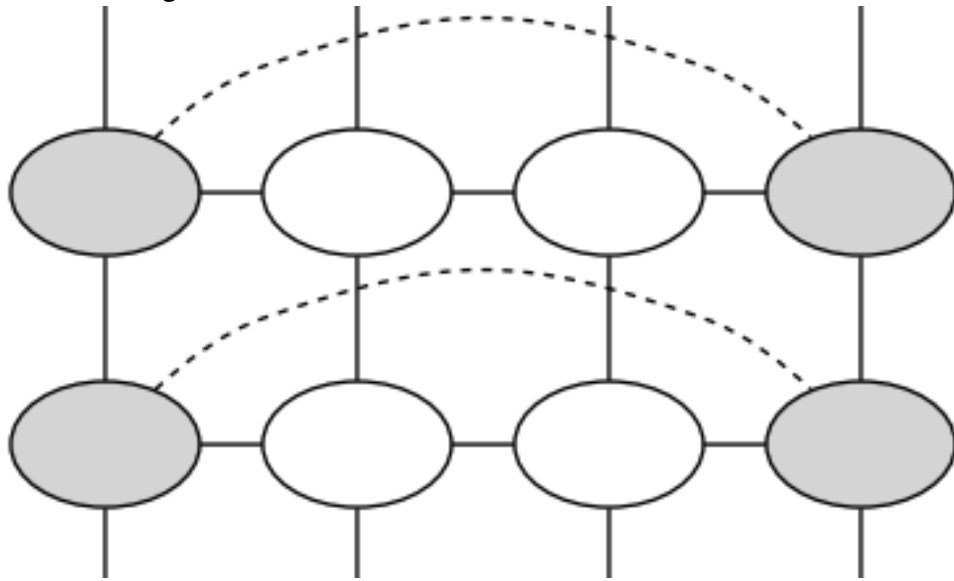


Fonte – (??)

3.4.1 Emparelhamento

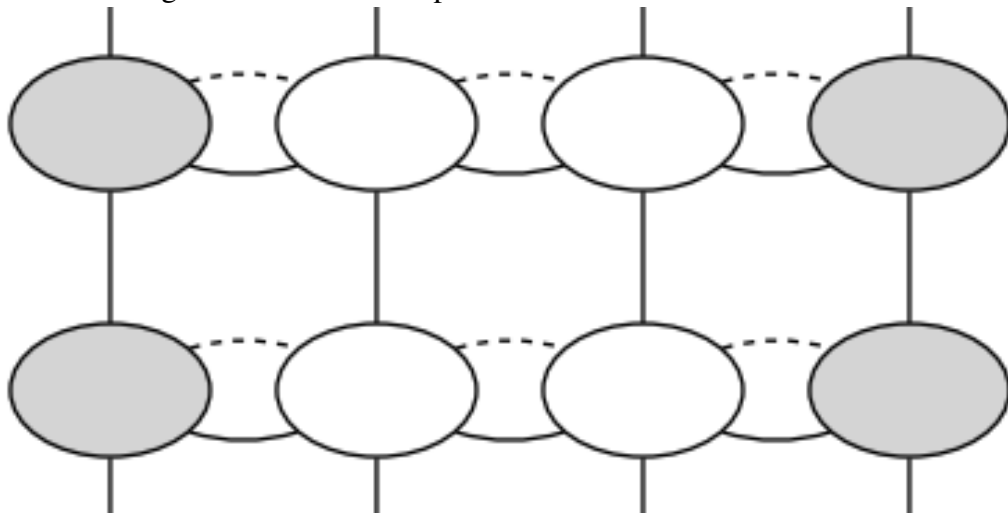
O aspecto mais relevante do Problema do Carteiro Chinês é encontrar a melhor associação possível entre os vértices de grau ímpar, ou seja, encontrar quais cópias de arestas devem ser adicionadas de maneira que o circuito euleriano tenha custo mínimo. A melhor associação possível entre os vértices é chamada de **emparelhamento perfeito** (ou *matching*

Figura 31 – Vértices associados de maneira não ótima



Fonte – Elaborado pelo autor.

Figura 32 – Arestas duplicadas entre os vértices associados



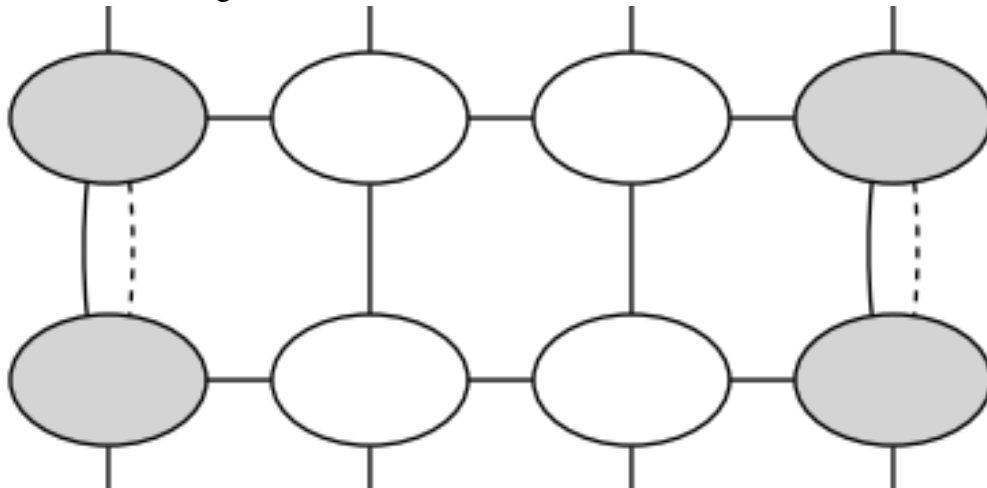
Fonte – Elaborado pelo autor.

perfeito) que é um problema NP-completo. Sua solução pode ser obtida através do algoritmo de Edmonds (??) ou utilizando modelos de PLI (??????????).

A rotina de emparelhamento máximo de Edmonds (**Blossom I**) (??) tem complexidade computacional na ordem de $O(n^3)$ (??) operações aritméticas, portanto, o algoritmo para encontrar um circuito euleriano tem complexidade, no pior caso, igual a $O(n^3)$.

A ideia básica do algoritmo Blossom I é partindo de um matching M vazio, construir um matching máximo iterativamente através de caminhos aumentados no grafo. Neste algoritmo, uma arborescência (**blossom**), é um ciclo de tamanho ímpar no grafo, ele é contraído em um único vértice, com a busca continuando no grafo contraído. O matching máximo é encontrado

Figura 33 – Vértices associados de maneira ótima



Fonte – Elaborado pelo autor.

quando nenhum caminho aumentado no grafo é encontrado.

O Algoritmo 5 é justamente o Blossom I mencionado. Já existem algoritmos que calculam o matching máximo de um grafo de uma maneira mais eficiente que a proposta por Edmonds, como é o caso do Blossom V (??).

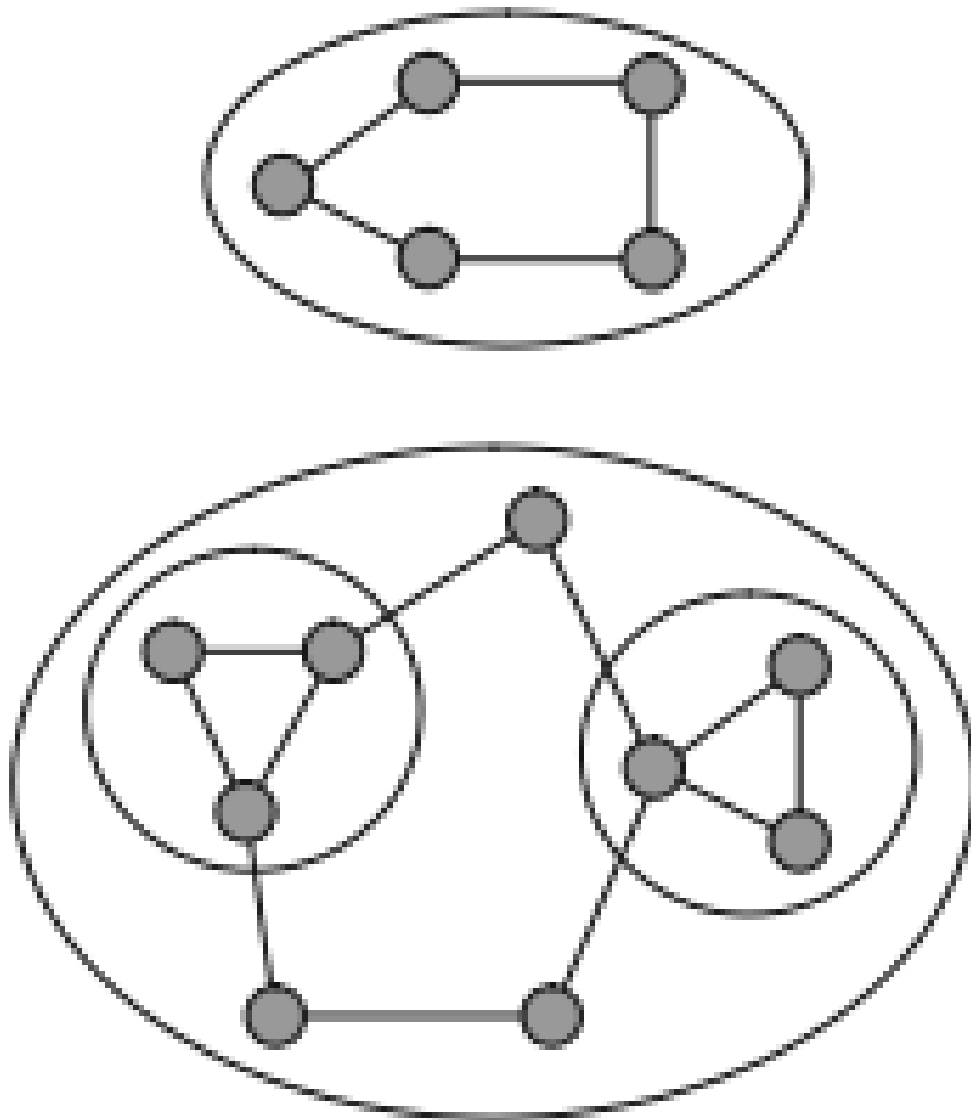
O Algoritmo 23 demonstra como encontrar um caminho aumentado de um grafo.

3.4.2 Tipos de Problemas do Carteiro Chinês

De maneira geral, os Problemas do Carteiro Chinês podem ser resolvidos em grafos conexos seguindo estes passos descritos a seguir:

1. Dado um grafo $G = (N, L)$, fortemente conexo, onde a cada aresta $e \in E$ é associado um custo d_e ;
2. Identifique o conjunto $I \subseteq N$ de vértices de grau ímpar (ou com grau de entrada diferente do grau de saída). Se $I = \emptyset$ então G é euleriano. Faça $G_e = G$, e vá ao passo 6;
3. Construa o grafo completo $G_I = (I, C)$, onde C é o conjunto de caminhos mínimos entre todos os pares de vértices de I , calculados no grafo G ;
4. Encontre o emparelhamento de custo mínimo (emparelhamento perfeito) em G_I ;
5. Identifique os caminhos mínimos $C^* \subseteq C$ em G , os quais ligam os vértices do emparelhamento de mínimo custo. Seja $E^* \subseteq E$ o conjunto de todas as arestas (arcos) que formam os caminhos mínimos em C^* . O grafo aumentado $G_e = (N, E \cup E^*)$ é euleriano;
6. Construa um circuito euleriano em G_e utilizando o algoritmo de Fleury (??) ou o algoritmo de Hierholzer (??).

Figura 34 – Exemplos de Blossoms



Fonte – (??)

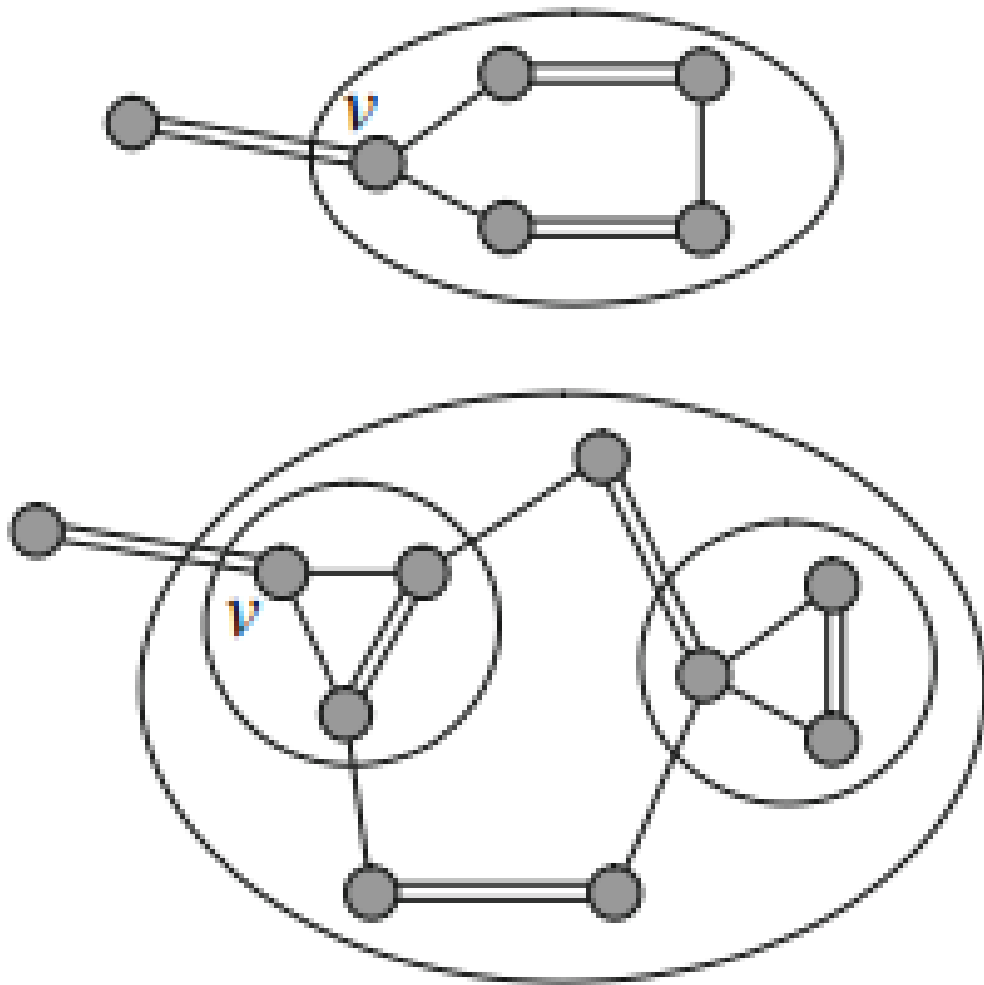
A maior diferença entre maneiras de resolver o PCC ocorre justamente no passo 4, pois dependendo do tipo de PCC tratado, o emparelhamento perfeito é calculado de maneiras distintas.

3.4.2.1 Problema do Carteiro Chinês Não Dirigido (PCCND)

O Problema do Carteiro Chinês Não Dirigido (PCCND) possui $G = (N, A, E)$ tal que $A = \emptyset$, ou seja G não possui arcos, consequentemente todos os seus links são não dirigidos (arestas). A Figura 36 mostra um exemplo de grafo em que pode-se aplicar o PCCND.

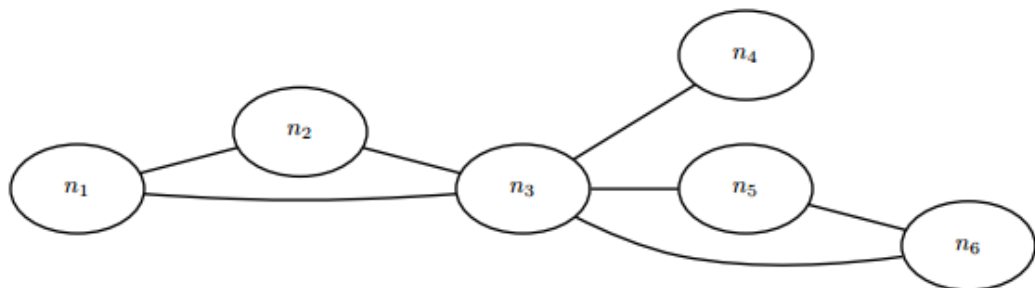
Essa formulação de PCC é a mais simples e remete a formulação inicial proposta

Figura 35 – Possível emparelhamento máximo sobre os blossoms da Figura 34



Fonte – (??)

Figura 36 – Exemplo de Grafo Não Dirigido



Fonte – (??)

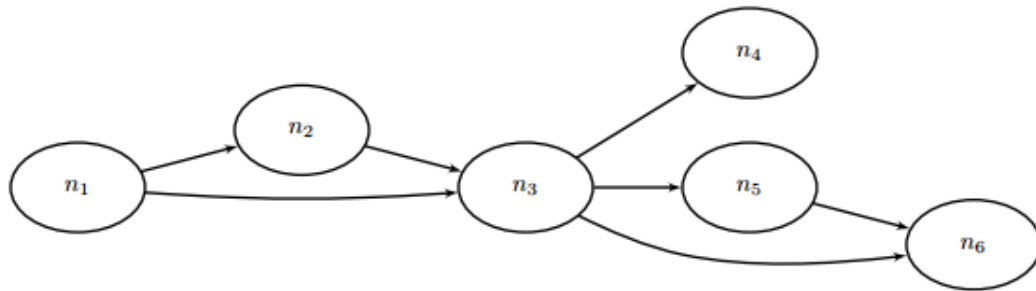
por Kwan (??). Este tipo de formulação pode ser resolvida utilizando o algoritmo proposto por Edmonds (??), que resolve o emparelhamento perfeito em tempo polinomial $O(n^3)$.

A condição necessária e suficiente para existência de um circuito euleriano em um grafo não dirigido é que o grafo seja fortemente conexo e todos os vértices possuam grau par.

3.4.2.2 Problema do Carteiro Chinês Dirigido (PCCD)

O Problema do Carteiro Chinês Dirigido (PCCD) é o tipo de PCC em que $G = (N, A, E)$ tal que $E = \emptyset$, ou seja G não possui arestas, todos os seus links são dirigidos (arcos). A Figura 37 mostra um exemplo de grafo em que pode-se aplicar o PCCD.

Figura 37 – Exemplo de Grafo Dirigido



Fonte – (??)

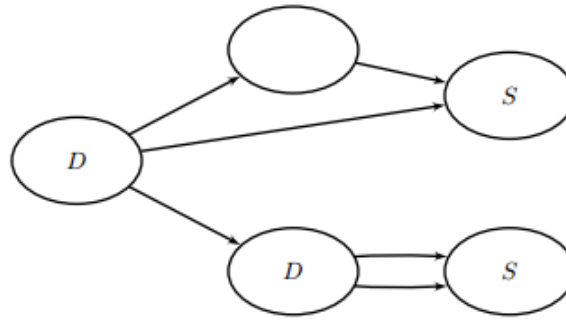
A condição necessária e suficiente para existência de um circuito euleriano em um grafo dirigido é que o grafo em questão seja fortemente conexo e seja também simétrico. Quando em alguns vértices o número de arcos de entrada diverge do número de arcos de saída, o grafo não é unicursal e, para torná-lo assim é necessário acréscimo de cópias apropriadas de alguns arcos.

Beltrami e Bodin (??) resolvem o emparelhamento perfeito resolvendo um simples Problema de Transporte. Nesse caso, os vértices com excesso de entrada serão considerados como suprimento e os com excesso de saída como demanda. A solução do Problema de Transporte indica qual nó de suprimento deva ser associado a qual de demanda. As cópias dos arcos escolhidos pelo modelo devem ser acrescentadas ao grafo, ao longo dos caminhos mínimos que ligam os vértices de suprimento aos de demanda na solução do Problema de Transporte. A Figura 38 mostra uma representação do grafo de suprimento e demanda de acordo com o número de arcos que chegam ou saem de um determinado vértice. Os vértices rotulados com S se comportarão como suprimento na modelagem e os rotulados com D se comportarão como demanda. Neste exemplo o vértice que não está rotulado não entrará no cálculo do emparelhamento pois seu grau de entrada é igual ao grau de saída.

A Tabela 0 mostra outras técnicas utilizadas na literatura para resolver o emparelhamento perfeito para o PCCD.

Em (??) m corresponde ao número de arcos e n ao número de vértices do grafo. Em

Figura 38 – Exemplo de Grafo aplicado ao Problema de Transportes



Fonte – Elaborado pelo autor.

Tabela 0 – Formas de resolver o PCCD

Autor(es)	Técnica	Complexidade
Beltrami e Bodin (??)	Problema de Transportes	$O(mn^2)$
Edmonds (??)	Fluxo de Custo Mínimo	$O(n^3)$
Lin e Zhao (??)	Teorema de Folga Complementar	$O(kn^2)$

Fonte – Elaborado pelo autor

(??), n também corresponde ao número de vértices do grafo. Por fim, em (??), n corresponde ao número de vértices do grafo e k depende da estrutura do grafo. Lin e Zhao mostraram que k é menor que m e n para grafos esparsos, defendendo assim seu uso para estes tipos de grafos.

3.4.2.3 Problema do Carteiro Chinês Misto (PCCM)

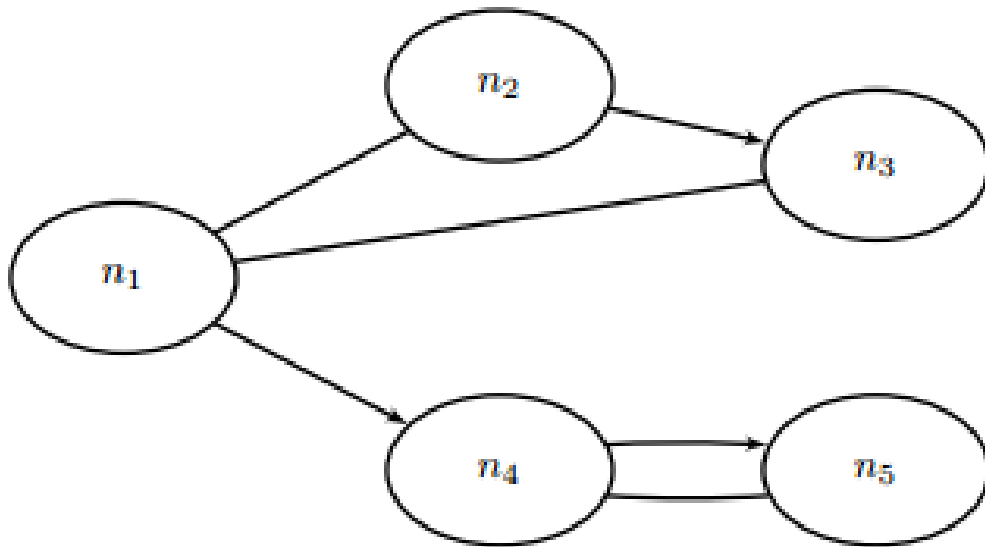
O Problema do Carteiro Chinês Misto (PCCM) possui $G = (N, A, E)$ tal que $A \neq \emptyset$ e $E \neq \emptyset$, ou seja G possui tanto arcos quanto arestas. Calcular o PCCM consiste em achar um circuito de custo mínimo em G , contendo todos os arcos em A e todas as arestas em E . A Figura 39 mostra um exemplo de grafo em que pode-se aplicar o PCCM.

As condições necessárias e suficientes para a unicursalidade de um grafo misto são (??):

- G deve ser fortemente conexo;
- Em cada vértice deve incidir um número par de links;
- Para qualquer $X \subset N$, a diferença entre o número de arcos orientados de X a NX e o número de arcos orientados de NX a X deve ser menor, ou igual, ao número de arestas que conectam X e NX

Papadimitriou (??) demonstrou que o PCCM é NP-completo. Portanto, não existem soluções exatas com complexidade polinomial para este caso.

Figura 39 – Exemplo de Grafo Misto



Fonte – (??)

Uma das maneiras de se calcular um PCCM é transformar o grafo fornecido em um grafo dirigido (??) , transformando as arestas em arcos contrariamente direcionados e após este passo resolver o PCCD equivalente.

Com esta modificação no grafo G , os passos descritos anteriormente para calcular genericamente qualquer PCC sofrem uma pequena alteração:

1. Dado um grafo $G = (N, A, E)$, fortemente conexo, onde a cada aresta $e \in E$ é associado um custo d_e ;
2. Para cada $e_i \in E$, criar dois arcos contrariamente direcionados ligando os mesmos vértices que e_i liga e com o mesmo custo d_{e_i} e remover e_i do grafo;
3. Identifique o conjunto $I \subseteq N$ de vértices com grau de entrada diferente do grau de saída. Se $I = \emptyset$ então G é euleriano. Faça $G_e = G$, e vá ao passo 7;
4. Construa o grafo completo $G_I = (I, C)$, onde C é o conjunto de caminhos mínimos entre todos os pares de vértices de I , calculados no grafo G ;
5. Encontre o emparelhamento de custo mínimo (emparelhamento perfeito em G_I);
6. Identifique os caminhos mínimos $C^* \subseteq C$ em G , os quais ligam os vértices do emparelhamento de mínimo custo. Seja $E^* \subseteq E$ o conjunto de todos os arcos que formam os caminhos mínimos em C^* . O grafo aumentado $G_e = (N, E \cup E^*)$ é euleriano;
7. Construa um circuito euleriando em G_e utilizando o algoritmo de Fleury (??) ou o algoritmo de Hierholzer (??).

3.5 ALGORITMOS DE CAMINHO MÍNIMO

Como os algoritmos para calcular os PCC's fazem uso de algoritmos de caminhos mínimos no cálculo do emparelhamento perfeito se faz importante destinar uma seção para falar sobre estes algoritmos. Os algoritmos tratados nesta seção são justamente os algoritmos mais conhecidos e mais utilizados na literatura para calcular caminhos mínimos em grafos, são eles: algoritmo de Dijkstra e algoritmo de Bellman-Ford.

3.5.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra (??) é um algoritmo guloso eficiente bastante conhecido para calcular caminhos mínimos entre vértices de um grafo dirigido ou não dirigido. É importante salientar que para que o algoritmo de Dijkstra funcione o grafo deve ser conexo e os custos dos links devem ser positivos, como o PCC também necessita destas condições, este algoritmo pode ser utilizado sem restrições para a descoberta do emparelhamento seja qual for o PCC utilizado.

O algoritmo inicia no vértice designado como vértice inicial $n_{inicial}$, e este tem obviamente distância zero para ele mesmo. A partir $n_{inicial}$ expande-se uma árvore com todos os vértices alcançáveis a partir de $n_{inicial}$. Os vértices desta árvore são visitados por ordem de proximidade de $n_{inicial}$, ou seja, o primeiro vértice visitado é o que está mais próximo de $n_{inicial}$. Os vértices que vão sendo visitados vão tendo sua própria árvore expandida e o algoritmo procura a menor distância checando se é possível relaxar a distância entre os vértices através da desigualdade triangular:

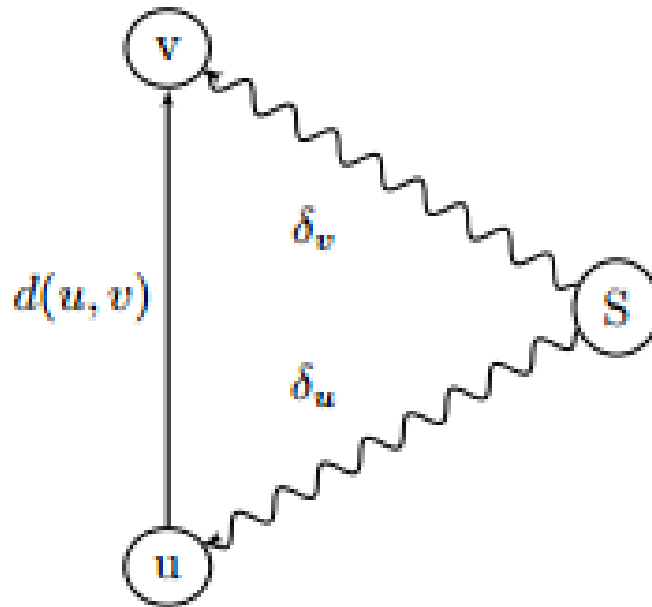
$$\delta_v \geq \delta_u + d(u, v)$$

A Figura 40 demonstra exatamente esta desigualdade triangular.

Nesta desigualdade, δ_v representa a distância de $n_{inicial}$ até o vértice v , δ_u se refere a distância de $n_{inicial}$ até o vértice u e $d(u, v)$ representa a distância entre os vértices u e v . Essa desigualdade checa se a distância do vértice $n_{inicial}$ até o vértice v é menor passando pelo vértice u . Se confirmado que a distância é menor se passar pelo vértice u a distância de $n_{inicial}$ até v é atualizada. O Algoritmo 6 descreve em todos os detalhes o algoritmo de Dijkstra. Os Algoritmos 7 e 8 são auxiliares ao 6 e foram separados somente para um melhor entendimento.

O algoritmo de Dijkstra tem, em sua implementação mais eficiente, complexidade de

Figura 40 – Relaxação através da desigualdade triangular



Fonte – Elaborado pelo autor.

$O(|E| \log_2 |N|)$, onde $|E|$ corresponde ao número de arestas (arcos) do grafo e $|N|$ corresponde ao número de vértices dele.

3.5.2 Algoritmo de Bellman-Ford

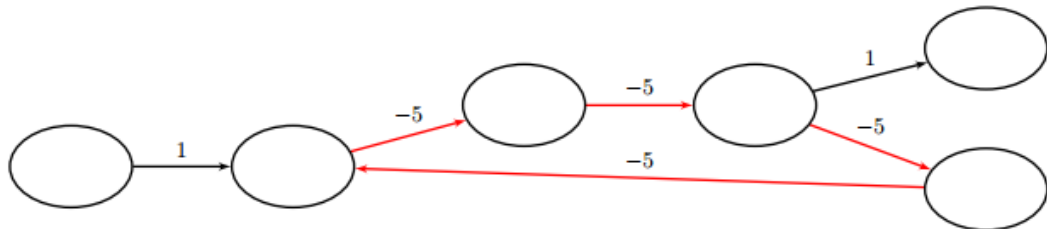
O algoritmo de Bellman-Ford (??????) resolve o problema de descobrir, a partir de um vértice inicial, o caminho mais curto para todos os outros vértices de um grafo. Diferentemente do algoritmo de Dijkstra, o algoritmo de Bellman-Ford não impõe nenhuma restrição com relação ao sinal das distâncias dos vértices, o que torna uma solução mais genérica que a solução fornecida por Dijkstra e capacita o uso do Bellman-Ford em situações onde os custos de arestas podem ser negativos.

A ideia básica do algoritmo de Bellman-Ford é realizar o relaxamento das arestas de uma maneira inteligente. A relaxação das arestas neste caso é semelhante ao Algoritmo 7 visto anteriormente. Este relaxamento é realizado para todas as arestas de um grafo G , $|N[G]| - 1$ vezes, onde $|N[G]|$ é o número de vértices do grafo deste grafo.

Um empecilho à execução correta do algoritmo de Bellman-Ford em um grafo é a existência de ciclos negativos neste grafo. Pois se houver um ciclo negativo, sempre entrar neste ciclo será a opção que dará o menor caminho possível, consequentemente o algoritmo não funcionará de maneira adequada. Um ciclo negativo pode ser encontrado em um grafo se

for possível ainda relaxar alguma aresta do grafo, mesmo já tendo sido realizadas $|N[G]| - 1$ tentativas de relaxamento das arestas. O Algoritmo 14 descreve esta checagem de ciclos negativos e a Figura 41 mostra um exemplo de ciclo negativo em um grafo.

Figura 41 – Exemplo de grafo com ciclo negativo destacado em vermelho



Fonte – Elaborado pelo autor.

O algoritmo de Bellman-Ford é descrito no Algoritmo 9.

Apesar do algoritmo de Bellman-Ford ser mais versátil que o algoritmo de Dijkstra, ele é mais lento. Sua complexidade é $O(|N||E|)$, onde $|N|$ corresponde ao número de vértices do grafo e $|E|$ corresponde ao número de arestas (arcos) dele. O fato da técnica de Dijkstra ser mais rápida que a de Bellman-Ford faz com que, em geral, este último só seja utilizado nos casos específicos que o grafo tem custos negativos.

4 TRABALHOS RELACIONADOS

Os trabalhos relacionados ao Problema do Carteiro Chinês dividem-se em dois grupos distintos: os que procuram por uma solução exata para o problema e os que não garantem que a solução exata é encontrada, pois estes utilizam meta-heurísticas. Comumente são utilizadas soluções exatas para os problemas do Carteiro Chinês Dirigido e não Dirigido, já para o caso misto, por ser um problema NP-Completo, geralmente são utilizadas soluções aproximadas.

4.1 PROBLEMA DO CARTEIRO CHINÊS COM SOLUÇÕES EXATAS

4.1.1 Problema do Carteiro Chinês: escolha de métodos de solução

Este trabalho (??) apresenta três modelos de PLI para calcular os três tipos de PCC (dirigido, não dirigido e misto). Além disso é apresentado um algoritmo que auxilia na escolha de uma abordagem adequada para a resolução do PCC.

Os modelos de PLI apresentados calculam o emparelhamento perfeito do grafo, após o cálculo do emparelhamento o problema é facilitado pois se resume a encontrar um circuito euleriano em um grafo que já é euleriano.

O primeiro modelo proposto neste trabalho tem o intuito para determinar o emparelhamento perfeito em grafos não dirigidos. Este modelo é uma versão modificada do modelo proposto por Baker (??) .

Modelo de PLI 1: Determinação de emparelhamento perfeito para grafos não dirigidos.

$$\begin{aligned}
 & \text{Min } \sum_{i,j \in E} l_{ij} x_{ij} \\
 & \text{Sujeito a :} \\
 & \sum_{j:(i,j) \in E} x_{ij} = \sum_{j:(j,i) \in E} x_{ji} , \forall i \\
 & x_{ij} + x_{ji} \geq 1 , \forall (i,j) \in E \\
 & x_{ij} \in [0, 1] , \forall (i,j) \in E
 \end{aligned}$$

Ainda dentro desta discussão sobre a resolução do PCC em grafos não orientados, existe na literatura um número grande de variantes, como, por exemplo, Frederickson (??), os quais trabalham com o PCC para o caso de múltiplos veículos.

Para resolver os casos em que o grafo é dirigido este trabalho adota outro modelo de PLI baseado em Baker (??).

Modelo de PLI 2: Determinação de emparelhamento perfeito para grafos dirigidos.

$$\text{Min } \sum_{i,j \in E} l_{ij} x_{ij}$$

Sujeito a :

$$\sum_{j:(i,j) \in E} x_{ij} = \sum_{j:(j,i) \in E} x_{ji}, \forall i$$

$$x_{ij} \geq 1, \forall (i,j) \in E$$

Um detalhe sobre o modelo 2 é que, a sua matriz de coeficientes possui a propriedade da unimodularidade. Por este motivo, e segundo demonstrado por Ahuja (??), não é necessário considerar x_{ij} como sendo uma variável inteira e pode-se utilizar modelos de programação linear como o simplex, reduzindo assim a complexidade do problema.

Para grafos dirigidos a literatura apresenta uma série de extensões do PCC básico:

- Wang & Wen (??) trabalham com o PCC para grafos dirigidos com restrições de tempo;
- Lin & Zhao (??) desenvolvem um algoritmo para o PCC dirigidos, o qual também pode ser utilizado para o caso de múltiplos veículos (m-PCC);
- Ghiani & Improta (??) trazem algoritmos para resolver o problema do PCC hierárquico, tanto para redes dirigidas como para não dirigidas.

Para os casos em que o grafo é misto, este trabalho utilizou a modificação sugerida por Ahuja et al. (??). Este modelo inicialmente separa os arcos dirigidos e não dirigidos em dois conjuntos A e A'.

Modelo de PLI 3: Modelo de fluxo de custo mínimo para grafos mistos.

$$\text{Min} \sum_{i,j \in A} l_{ij}x_{ij} + \sum_{i,j \in A'} l_{ij}x_{ij}$$

Sujeito a :

$$\sum_{j:(i,j) \in A} x_{ij} + \sum_{j:(i,j) \in A'} x_{ij} = \sum_{j:(j,i) \in A} x_{ji} + \sum_{j:(j,i) \in A'} x_{ji}, \forall i$$

$$x_{ij} + x_{ji} \geq 1, \forall (i,j) \in A'$$

$$x_{ij} \geq 1, \forall (i,j) \in A$$

$$x_{ij} \in [0, 1], \forall (i,j) \in A$$

$$x_{ij} \in [0, 1], \forall (i,j) \in A'$$

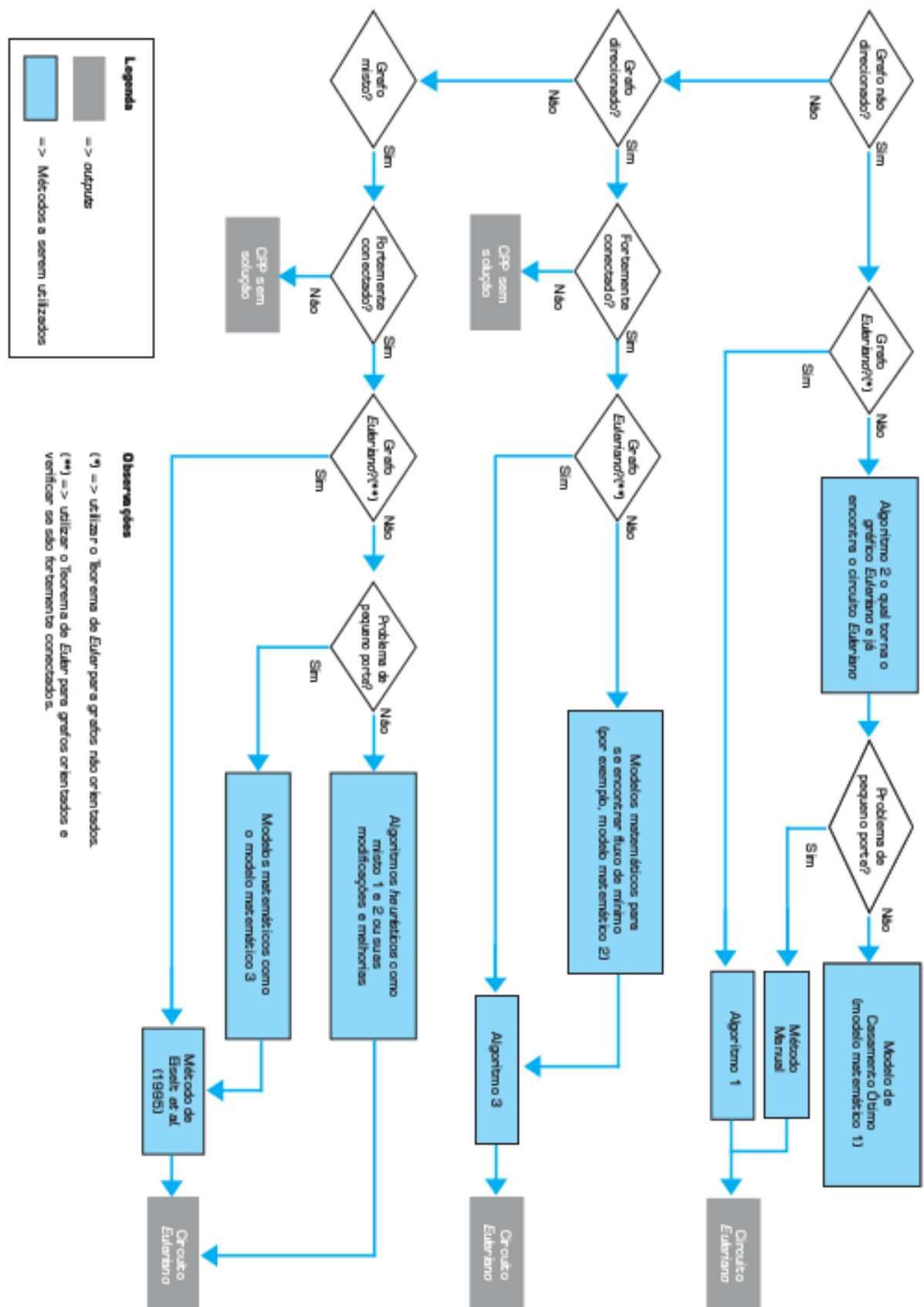
Este trabalho apresenta ainda um algoritmo que auxilia na escolha de uma abordagem adequada para a resolução do PCC. A solução do PCC pode ser resumida em três passos, independentemente do grafo ser não direcionado, direcionado ou misto.

1. Verificar se o grafo é Euleriano. Caso a resposta seja afirmativa, então, vá para o terceiro passo; caso contrário, vá para o segundo passo.
2. Obter um grafo Euleriano. Após este passo, deve-se ir para o terceiro passo.
3. Obter o circuito ou caminho Euleriano a partir do grafo Euleriano.

O algoritmo proposto se baseia em alguns pontos que são fundamentais para a escolha de um método de solução para o CPP. Entre estes pontos podem ser incluídas orientação do grafo (não direcionado, direcionado ou misto), conectividade do grafo (conexos, desconexos), grafo inicialmente euleriano ou não, número de arcos e nós do grafo estudado. Além disso, para a escolha de um método de solução devem ser levadas em consideração características relacionadas à complexidade e ao objetivo das soluções. As soluções podem ser algoritmos e teoremas para determinar se o grafo é ou não euleriano, podem ser soluções matemáticas simples e manuais para se encontrar o grafo euleriano com mínimo custo, podem ainda ser algoritmos com complexidade até no máximo $O(n^3)$ ou modelos matemáticos baseados em programação inteira para se encontrar o grafo euleriano com mínimo custo.

A Figura 42 mostra um fluxograma do algoritmo que auxilia a escolha de uma abordagem para resolução do PCC.

Figura 42 – Algoritmo que auxilia na escolha de uma abordagem adequada para a resolução do PCC



4.1.2 O problema do carteiro chinês, algoritmos exatos e um ambiente MVI para análise de suas instâncias: sistema XNÊS

Este trabalho (??) apresenta um estudo geral sobre o Problema do Carteiro Chinês (PCC), nas versões não dirigido, dirigido e misto. Nele também é apresentada uma ferramenta de Modelagem Visual Interativa (MVI), através do software (Sistema XNÊS) idealizado para gerar grafos instâncias e soluções exatas para as versões do Problema do Carteiro Chinês (PCC).

Para o PCC não dirigido, foi descrito neste trabalho, o seguinte modelo de PLI (??):

Modelo de PLI 1: Determinação de emparelhamento perfeito para grafos não dirigidos.

$$\text{Min } \sum_{i,j \in E} l_{ij} x_{ij}$$

Sujeito a :

$$\sum_{(n_i, n_j) \in E(S)} x_{ij} \geq 1, S \subset N$$

$$x_{ij} \in \mathbb{Z} > 0, \forall (n_i, n_j) \in E(S)$$

No modelo descrito acima, S é o conjunto de vértices com grau ímpar e $E(S) = \{(n_i, n_j) \mid n_i \in S, n_j \in N - S \text{ ou } n_i \in N - S, n_j \in S\}$.

Para resolver o PCC não dirigido, o trabalho aplica a formulação definida como Problema Clássico de Transportes (??????).

O Problema Clássico dos Transportes divide os vértices do grafo em dois conjuntos. O primeiro conjunto I é o conjunto dos vértices n_i , onde $d^+(n_i) > d^-(n_i)$, estes são ligados a um novo vértice f denominado de fonte. O segundo conjunto J é o conjunto dos vértices n_j onde $d^-(n_j) > d^+(n_j)$, estes são ligados a um vértice s , o qual se denomina de sumidouro. Sendo $d^+(n_i)$ definido como o número de arcos saindo e $d^-(n_i)$ o número de arcos entrando em n_i . A formulação do PCC dirigido (??) é mostrada a seguir:

Modelo de PLI 2: Determinação de emparelhamento perfeito para grafos dirigidos

formulado como um problema de transportes.

$$\text{Min} \sum_{n_i \in I} \sum_{n_j \in J} l_{ij} x_{ij}$$

Sujeito a :

$$\sum_{n_j \in J} x_{ij} = d^+(n_i) - d^-(n_i), \forall n_i \in I$$

$$\sum_{n_i \in I} x_{ij} = d^-(n_j) - d^+(n_j), \forall n_j \in J$$

Para a resolução do PCC misto, adotou-se a solução de Kappauf (??), onde ele considera a seguinte situação, seja $\bar{G} = (N, A \cup E^+ \cup E^-)$ o grafo direcionado associado a um grafo G . Para cada $e \in E$ fazer $c_{e^+} = c_{e^-} = c_e$. Uma circulação não negativa x em \bar{G} será o vetor de incidência da rota do carteiro quando $x_a \geq 1 \forall a \in A$, e $\forall e \in E \ x_{e^+} + x_{e^-} \geq 1$.

Modelo de PLI 3: Determinação de emparelhamento perfeito para grafos mistos.

$$\text{Min} \sum_{\langle n_i, n_j \rangle \in A} l_{ij} x_{ij} + \sum_{(n_i, n_j) \in E^+} l_{ij} x_{e^+} + \sum_{(n_i, n_j) \in E^-} l_{ij} x_{e^-}$$

Sujeito a :

$$x(\bar{\delta}(\bar{n})) - x(\bar{\delta}(n)) = 0 \forall n \in N$$

$$x_a \geq 1 \forall a \in A$$

$$x_{e^+} + x_{e^-} \geq 1 \forall e \in E$$

4.2 PROBLEMA DO CARTEIRO CHINÊS COM SOLUÇÕES APROXIMADAS

Devido à complexidade de resolver o problema do carteiro chinês de maneira exata, vários trabalhos tentam resolver o problema de maneira aproximada utilizando meta-heurísticas, alguns destes trabalhos são citados a seguir.

4.2.1 Aplicação de um algoritmo genético para o problema do carteiro chinês em uma situação real de cobertura de arcos

Este trabalho (??) utilizou a meta-heurística conhecida como Algoritmo Genético (??) para resolver o PCC misto. As soluções encontradas com o algoritmo genético foram comparadas com as soluções exatas para as mesmas instâncias.

4.2.2 Otimização de rotas para a coleta do lixo doméstico: um tratamento GRASP do PCCM

Este trabalho (??) também faz uso de uma meta-heurística para resolver o PCC misto. A meta-heurística escolhida por Paes foi a GRASP (Greedy Randomized Adaptive Search Procedure) (??).

A ideia do GRASP baseia-se em duas fases: fase de construção e fase de busca local.

4.2.2.1 Fase de Construção

Neste trabalho, a fase de construção consiste basicamente em orientar as arestas do grafo para se obter um grafo direcionado, e utilizando um algoritmo para resolver o Problema de Fluxo de Custo Mínimo (PFCM) tornar o grafo simétrico.

Os passos desta fase ocorrem da seguinte maneira, inicialmente cria-se uma lista de arestas não orientadas (U), contendo todas as arestas do grafo ($U = E$). A cada passo da fase de construção, uma aresta é selecionada aleatoriamente e orientada. O conjunto de arestas orientadas será denotado por E_d . Para a execução, a partir de agora, $d(v)$ será a diferença entre o número de arcos e arestas orientadas que entram em v e o número de arcos e arestas orientadas que saem de v , ou seja, $d(v) = d(v)^- - d(v)^+$.

A função de avaliação gulosa $w(i, j)$ de determinada aresta (i, j) , representa a conveniência de orientá-la, de forma a aproximar o grafo de um grafo simétrico, pois dependendo do tipo de orientação adotada o grafo pode se tornar mais distante de um grafo simétrico. Nesta abordagem o valor de $w(i, j)$ para determinada aresta (i, j) será o seu peso.

Portanto, após todos os pesos $w(i, j)$ terem sido calculados, o conjunto U será organizado em ordem decrescente de pesos w e a cada passo desta fase de construção uma aresta será escolhida entre os elementos da lista de candidatos restrita do GRASP, repetindo este processo até que todas as arestas tiverem sido orientadas e o grafo G_s seja simétrico.

4.2.2.2 Fase de Busca Local

A fase de busca local descrita neste trabalho consiste em identificar em G_s se existem mais de dois pares de arcos orientados em sentidos opostos, adjacentes aos mesmos vértices. Caso haja, o excesso de pares deve ser removido. Este procedimento reduz o custo do percurso do carteiro sem modificar as características do grafo.

4.2.3 Busca Tabu aplicada ao problema de roteirização de veículos

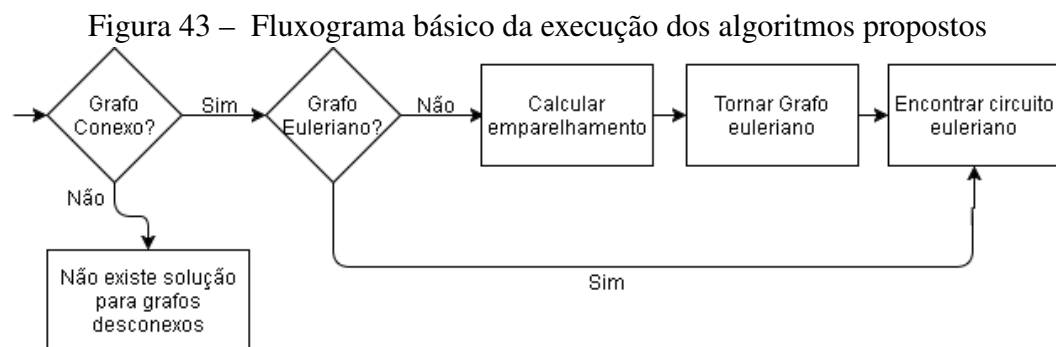
Em (??) e em (??) são abordados problemas de roteirização de veículos utilizando a meta-heurística Busca Tabu. No primeiro trabalho é mostrado uma implementação da Busca Tabu pra resolver problemas de roteirização de veículos e no segundo trabalho, outra a Busca Tabu é novamente implementada para resolver problemas de roteirização com janelas de tempo.

5 METODOLOGIA

Nesta seção serão descritos os três algoritmos propostos deste trabalho, um dos algoritmos resolve o PCC não dirigido, outro resolve o PCC dirigido e o último resolve o PCC misto. Um exemplo de execução será mostrado ao final de cada algoritmo. Todas as soluções obtidas são mostradas graficamente, utilizando o Framework Java Universal Network Graph (JUNG) (Java Universal Network Graph), que é largamente utilizadas em diversos estudos de grafos (?????). O algoritmo proposto mostra graficamente o carteiro percorrendo cada aresta do circuito euleriano descoberto.

Os três algoritmos diferem basicamente na maneira de calcular o emparelhamento. E o cálculo do emparelhamento é parte essencial destes três algoritmos, pois é a partir do resultado deste cálculo que se pode tornar o grafo euleriano, para poder assim encontrar um circuito euleriano nele.

A Figura 43 mostra um fluxograma com os passos básicos destes algoritmos para se encontrar o circuito euleriano.



Fonte – Elaborado pelo autor.

O Algoritmo 10 formaliza os passos tomados para resolução do PCC. Este algoritmo é descrito de maneira simplificadas.

A grande diferença entre os três algoritmos propostos é a forma que CALCULAR_EMPARELHAMENTO_PERFEITO é realizado. Todos os emparelhamentos são calculados utilizando modelos de Programação Linear Inteira (PLI), e a diferença entre eles é justamente a modelagem que é realizada. Na seção correspondente de cada algoritmo (PCC não dirigido, PCC dirigido e PCC misto) será mostrado qual modelo matemático foi utilizado em cada caso. O Algoritmo 11 gera um modelo matemático com sua função objetivo e um conjunto de restrições, além disso, ele executa o modelo matemático, que tem como resposta as arestas do

emparelhamento ótimo obtido.

O Algoritmo 12 utiliza o conjunto de arestas do emparelhamento ótimo obtidos no Algoritmo 11 e adiciona cópias destas arestas no grafo não euleriano para torná-lo euleriano.

O procedimento ENCONTRAR_CIRCUITO_EULERIANO encontra um circuito euleriano em um grafo euleriano utilizando o algoritmo de Fleury (Algoritmo 4, descrito na seção 3.3.1.2).

As seções posteriores descrevem como cada tipo de PCC é calculado neste trabalho.

5.1 PCC NÃO DIRIGIDO

Para o PCC não dirigido o algoritmo é exatamente o 10, se o grafo for desconexo, não existe solução para o PCC, e se o grafo já for um grafo euleriano, o problema passa a ser somente encontrar um circuito euleriano neste grafo, que o é feito no procedimento ENCONTRAR_CIRCUITO_EULERIANO.

O modelo de PLI 5.1 é utilizado para calcular o emparelhamento no PCC não dirigido. A ideia é colocar nas restrições deste modelo que o somatório das arestas saindo de cada vértice seja igual a um, com isso, o modelo irá escolher somente uma aresta de cada vértice para duplicar, tornando assim o grau dos vértices par e consequentemente o grafo se tornará euleriano.

Modelo de PLI 1: Determinação de emparelhamento perfeito para grafos não dirigidos.

$$\text{Min } \sum_{i,j \in E} l_{ij} x_{ij}$$

Sujeito a :

$$\sum_{(i,j) \in E} x_{ij} = 1, \forall i, \text{ com } i < j$$

$$x_{ij} \text{ int}, \forall (i,j) \in E$$

Se o grafo G for inicialmente não euleriano, um circuito euleriano em um grafo não dirigido pode ser encontrado seguindo os seguintes passos:

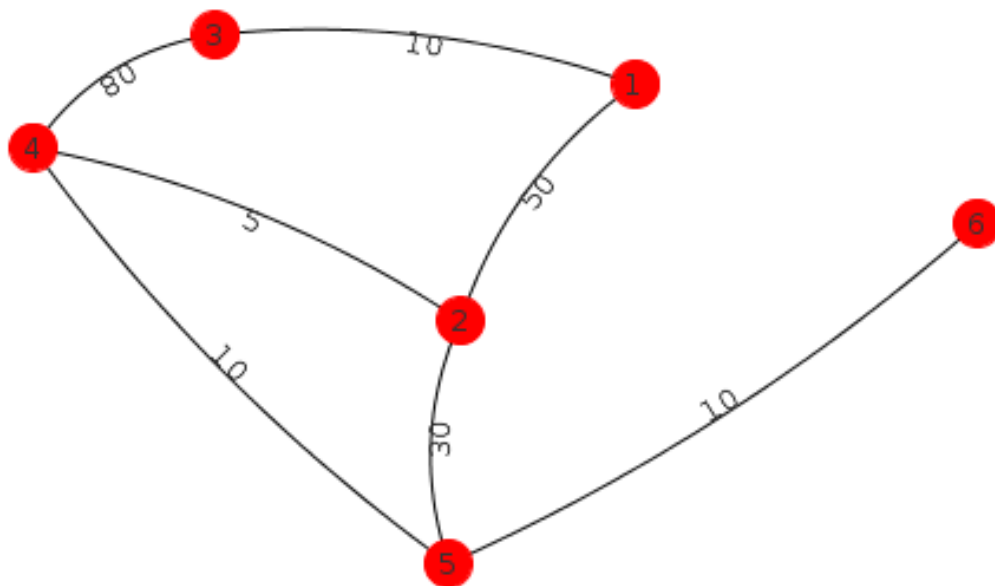
- Gerar um grafo G' que é uma cópia de G porém remover os vértices de grau par. O grafo G' terá m vértices;
- Calcular as menores distâncias entre estes m vértices em G utilizando o Algoritmo 6 de Dijkstra (??), descrito na Seção 3.5.1;

- Adicionar arestas em G' , de tal maneira que cada aresta adicionada tenha o custo igual a distância mínima calculada no item anterior. Após este procedimento obtém-se um grafo completo K_m ;
- Encontrar o emparelhamento ótimo entre estes vértices utilizando o Modelo de PLI 1;
- Adicionar estes $m/2$ caminhos mínimos, como arestas ligando os vértices do emparelhamento no grafo G . G após este passo é euleriano.
- Encontrar um circuito euleriano em G .

5.1.1 Execução do Algoritmo para o PCCND

O exemplo de execução do PCCND será com o grafo mostrado na Figura 44.

Figura 44 – Grafo exemplo PCCND



Fonte – Elaborado pelo autor.

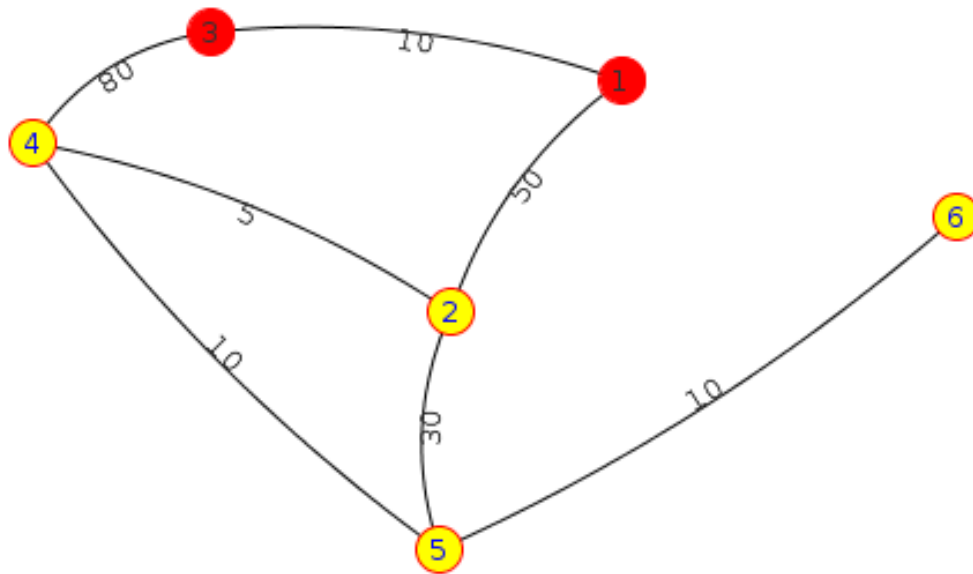
Como mostrado na Figura 45, os vértices 2, 4, 5 e 6 têm grau ímpar, portanto para execução do algoritmo serão seguidos os passos descritos anteriormente para o caso em que o grafo é inicialmente não euleriano.

O primeiro passo a seguir é gerar um grafo retirando os vértices de grau par que existiam. O estado do grafo gerado é mostrado na Figura 46.

No segundo passo são calculadas as distâncias mínimas entre estes vértices utilizando o algoritmo de Dijkstra. O Quadro 2 mostra o cálculo destas distâncias.

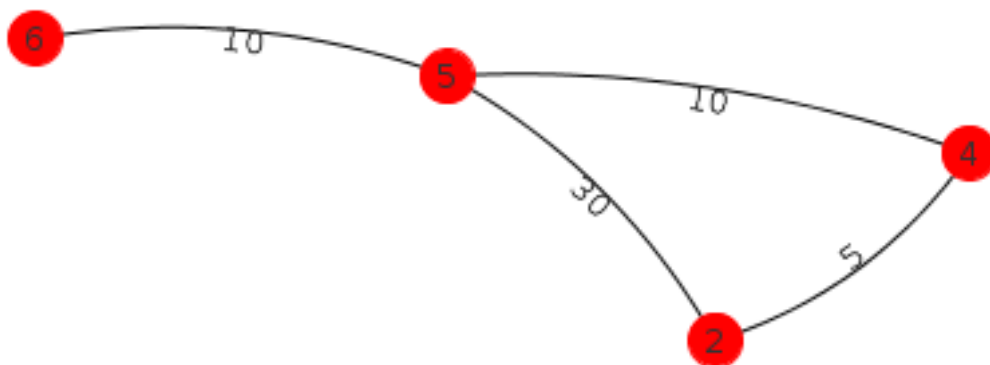
O próximo passo consiste em adicionar arestas no grafo, de tal maneira que cada

Figura 45 – Grafo para exemplo de execução do PCCND com vértices ímpares selecionados



Fonte – Elaborado pelo autor.

Figura 46 – Grafo para exemplo de execução do PCCND com vértices pares removidos



Fonte – Elaborado pelo autor.

Quadro 2 – Distâncias Mínimas exemplo execução PCCND

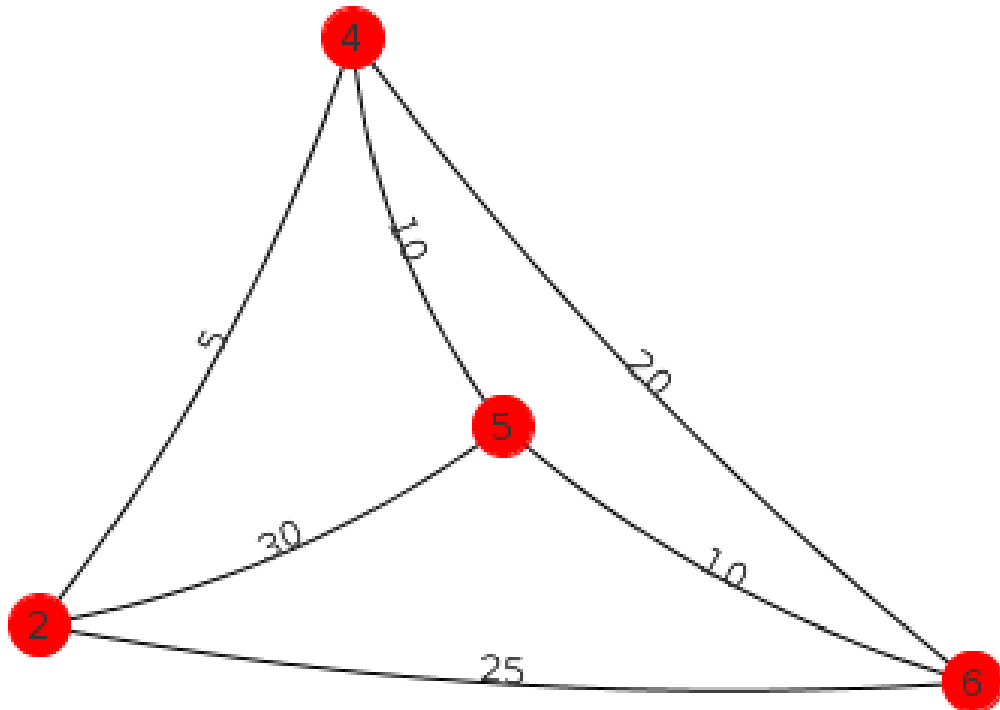
Vértice Origem	Vértice Destino	Distância Mínima	Caminho
2	4	5	2-4
2	5	15	2-4-5
2	6	25	2-4-5-6
4	5	10	4-5
4	6	20	4-5-6
5	6	10	5-6

Fonte – Elaborado pelo autor.

aresta adicionada tenha o custo igual a distância mínima calculada no passo anterior. O grafo após as adições de arestas é mostrado na Figura 47.

No próximo passo do algoritmo o modelo de PLI para calcular o emparelhamento ótimo de um grafo não dirigido deve ser gerado e executado. Para este exemplo, foi gerado o

Figura 47 – Grafo para exemplo de execução do PCCND após adicionadas cópias de arestas



Fonte – Elaborado pelo autor.

modelo 5.1.

$$\text{Minimizar } 25X_{26} + 5X_{24} + 10X_{56} + 10X_{45} + 20X_{46} + 30X_{25}$$

Sujeito a :

$$X_{24} + X_{45} + X_{46} = 1$$

$$X_{26} + X_{56} + X_{46} = 1$$

$$X_{56} + X_{45} + X_{25} = 1$$

$$X_{26} + X_{24} + X_{25} = 1$$

Ao ser executado, este modelo matemático dá a solução mostrada no Quadro 3 como solução ótima para o emparelhamento.

A informação contida neste quadro é que devemos duplicar as arestas do caminho mínimo entre os vértices 2 e 4 e entre os vértices 5 e 6. Como o menor caminho que liga o vértice 2 ao 4 é justamente a aresta (2,4), uma cópia desta aresta é adicionada ao grafo. Da mesma maneira, o menor caminho que liga o vértice 5 ao 6 é a aresta (5,6), portanto uma cópia desta aresta também é adicionada ao grafo. Com a adição destas cópias de arestas obtém-se um

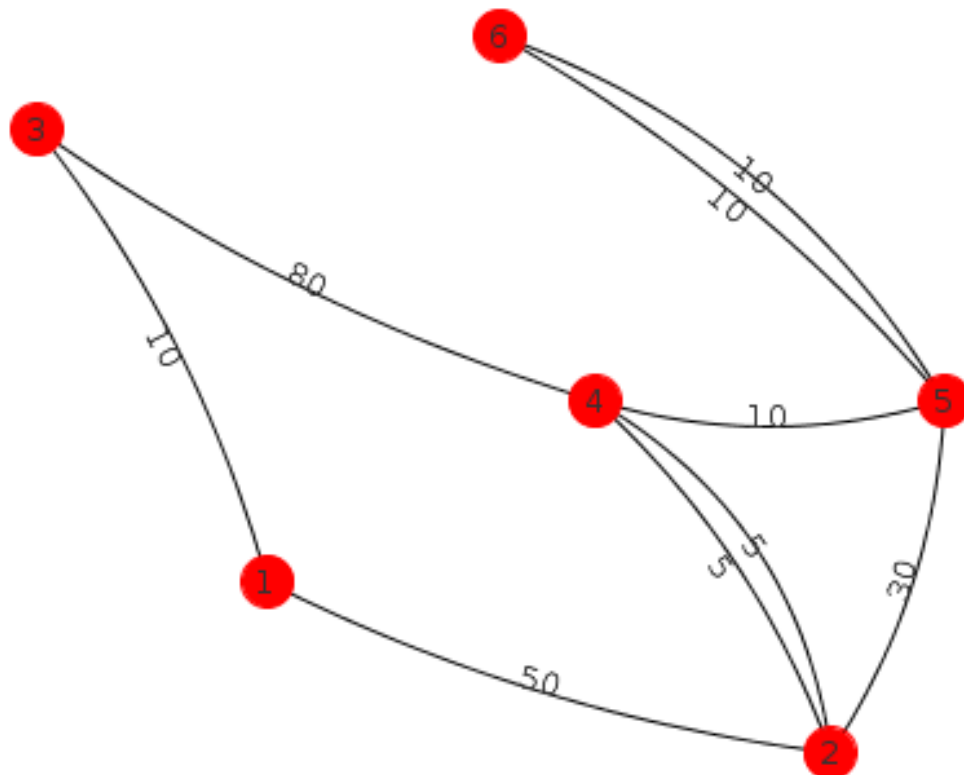
Quadro 3 – Resultado da execução do modelo de PLI 1 no exemplo de execução do PCCND

Variável	Valor
X24	1
X56	1
X25	0
X46	0
X26	0
X45	0

Fonte – Elaborado pelo autor.

grafo euleriano, pois os vértices que antes tinham grau ímpar, aumentaram seu grau em um e se tornaram par. O grafo euleriano resultante deste procedimento é mostrado na Figura 48.

Figura 48 – Grafo euleriano para exemplo de execução do PCCND após adicionadas cópias de arestas

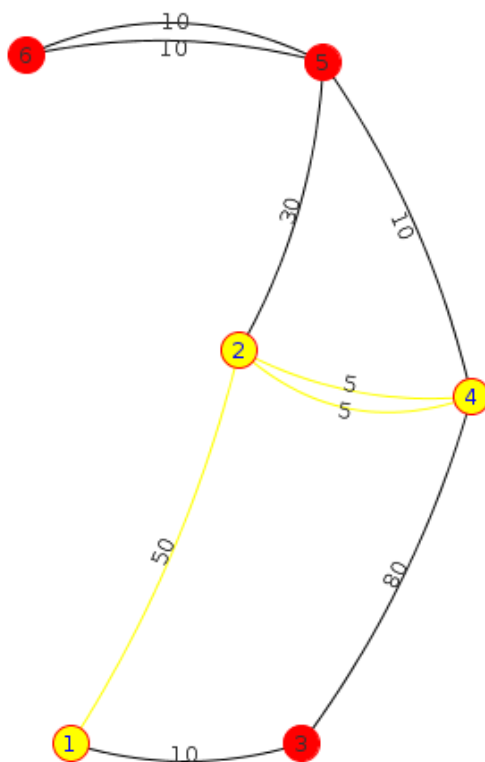


Fonte – Elaborado pelo autor.

O último passo do algoritmo consiste em encontrar um circuito euleriano no grafo . Foi utilizado o Algoritmo de Fleury (??) para, partindo do vértice 1, passar por todas as arestas do grafo e retornar ao vértice inicial. A Figura 49 mostra o grafo em um passo intermediário do algoritmo de Fleury, onde os vértices e arestas destacados em amarelo foram os que já foram visitados. E a Figura 50 mostra o grafo no final da execução, com todos os vértices e arestas

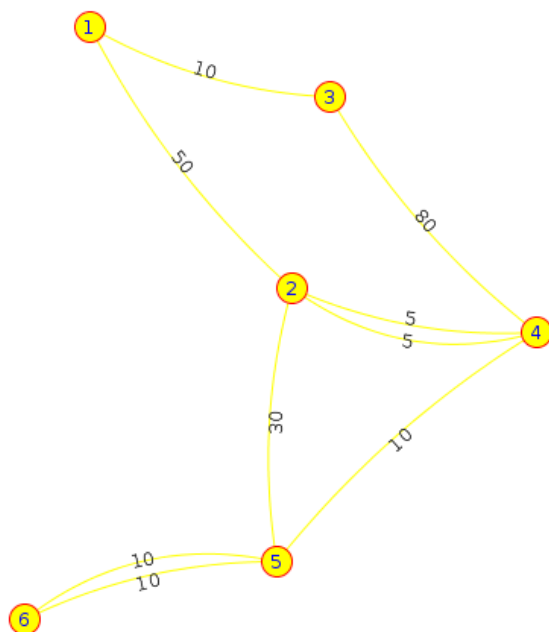
visitados.

Figura 49 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCND



Fonte – Elaborado pelo autor.

Figura 50 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCND



Fonte – Elaborado pelo autor.

O Quadro 4 mostra algumas informações desta execução do PCCND.

Quadro 4 – Informações da execução do exemplo de PCCND

Soma dos custos das arestas duplicadas	15
Número de arestas duplicadas	2
Arestas duplicadas	(2,4) e (5,6)
Soma do percurso do carteiro	210
Circuito euleriano	1-2-5-6-5-4-2-4-3-1

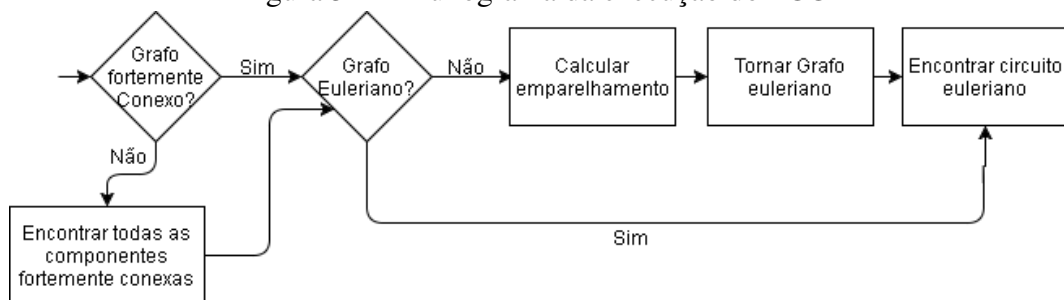
Fonte – Elaborado pelo autor.

5.2 PCC DIRIGIDO

O algoritmo para resolução do PCC dirigido também segue o Algoritmo 10. A diferença em relação ao PCC não dirigido está no modelo utilizado dentro da função `CALCULAR_EMPARELHAMENTO_PERFEITO`.

O fluxograma de execução deste dirigido é mostrado na Figura 51. É adicionado uma checagem para saber se o grafo é fortemente conexo, pois o PCCD só tem solução para grafos fortemente conexos. Se o grafo não for fortemente conexo, devem ser encontradas todas as componentes fortemente conexas dele e o algoritmo deve ser executado pra cada uma destas componentes.

Figura 51 – Fluxograma da execução do PCCD



Fonte – Elaborado pelo autor.

O modelo de PLI utilizado para calcular o emparelhamento no PCC dirigido força que o grau de entrada (deg^-), de cada vértice com grau ímpar, seja igual ao grau de saída dele (deg^+). Isto é obtido fazendo com que o somatório das variáveis que representam as arestas de cada vértice seja igual à diferença entre os graus de entrada e saída do vértice. O Modelo de PLI 2 descreve formalmente esta ideia.

Modelo de PLI 2: Determinação de emparelhamento perfeito para grafos dirigidos.

$$\text{Min } \sum_{i,j \in E} l_{ij} x_{ij}$$

Sujeito a :

$$\sum_{(i,j) \in N} x_{ij} = \deg^-(i) - \deg^+(i), \text{ se } \deg^-(i) > \deg^+(i)$$

$$\sum_{(i,j) \in N} x_{ji} = \deg^+(i) - \deg^-(i), \text{ se } \deg^+(i) > \deg^-(i)$$

$$x_{ij} \text{ int}, \forall (i,j) \in E$$

Se um grafo dirigido G for inicialmente não euleriano, um circuito euleriano pode ser encontrado seguindo os passos:

- Gerar um grafo G' que é uma cópia de G porém remover os vértices de grau par. O grafo G' terá m vértices;
- Calcular as menores distâncias entre estes m vértices em G utilizando o Algoritmo 6 de Dijkstra (??), descrito na Seção 3.5.1;
- Adicionar arestas em G' , de tal maneira que cada aresta adicionada tenha o custo igual a distância mínima calculada no item anterior. Após este procedimento obtém-se um grafo completo K_m ;
- Encontrar o emparelhamento ótimo entre estes vértices utilizando o Modelo 2;
- Adicionar estes $m/2$ caminhos mínimos, como arestas ligando os vértices do emparelhamento no grafo G . G após este passo é euleriano.
- Encontrar um circuito euleriano em G .

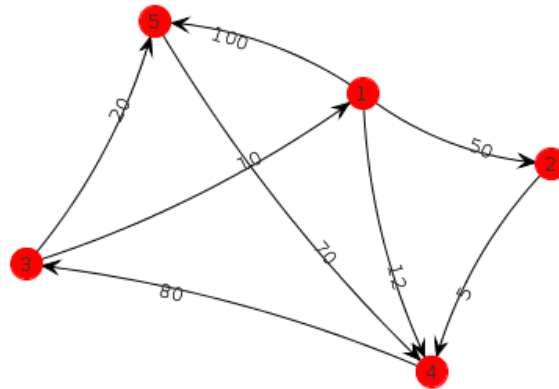
5.2.1 Execução do Algoritmo para o PCCD

O grafo como exemplo de execução do PCCD é mostrado na Figura 52.

Na Figura 53, os vértices 1, 3, 4 e 5 têm grau de entrada diferente do grau de saída, ou seja, todos estes vértices são desbalanceados, portanto este grafo não é euleriano e precisa se tornar euleriano para que a solução do PCCD seja encontrada. Esta execução do algoritmo, assim como no exemplo anterior, mostra os passos para se obter uma solução para o PCCD a partir de um grafo dirigido não euleriano.

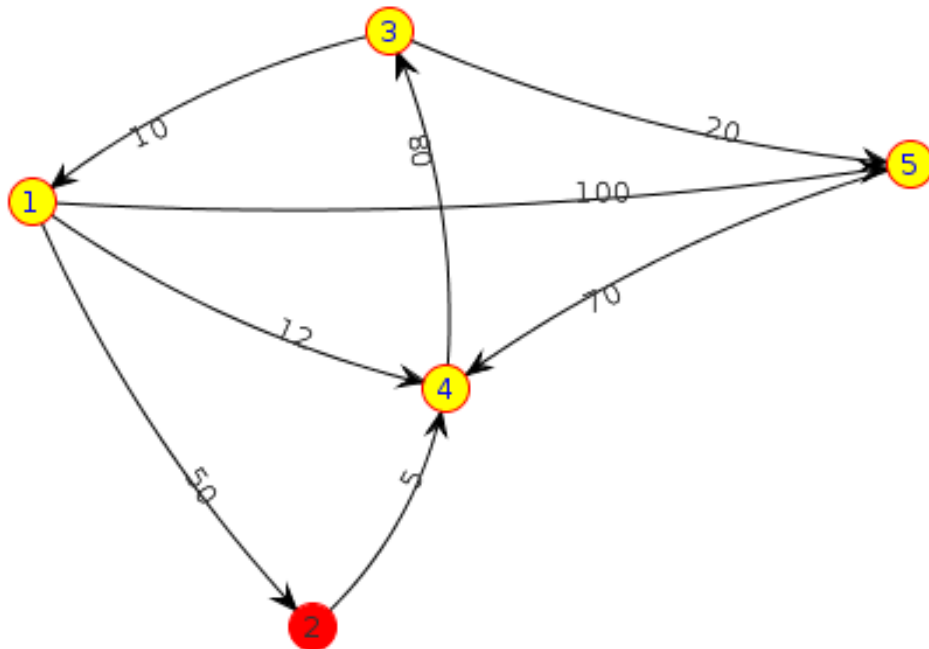
O primeiro passo da execução do algoritmo consiste em remover todos os vértices balanceados do grafo, pois o emparelhamento é realizado somente entre os vértices que possuem

Figura 52 – Grafo para exemplo de execução do PCCD



Fonte – Elaborado pelo autor.

Figura 53 – Grafo para exemplo de execução do PCCD com vértices desbalanceados selecionados

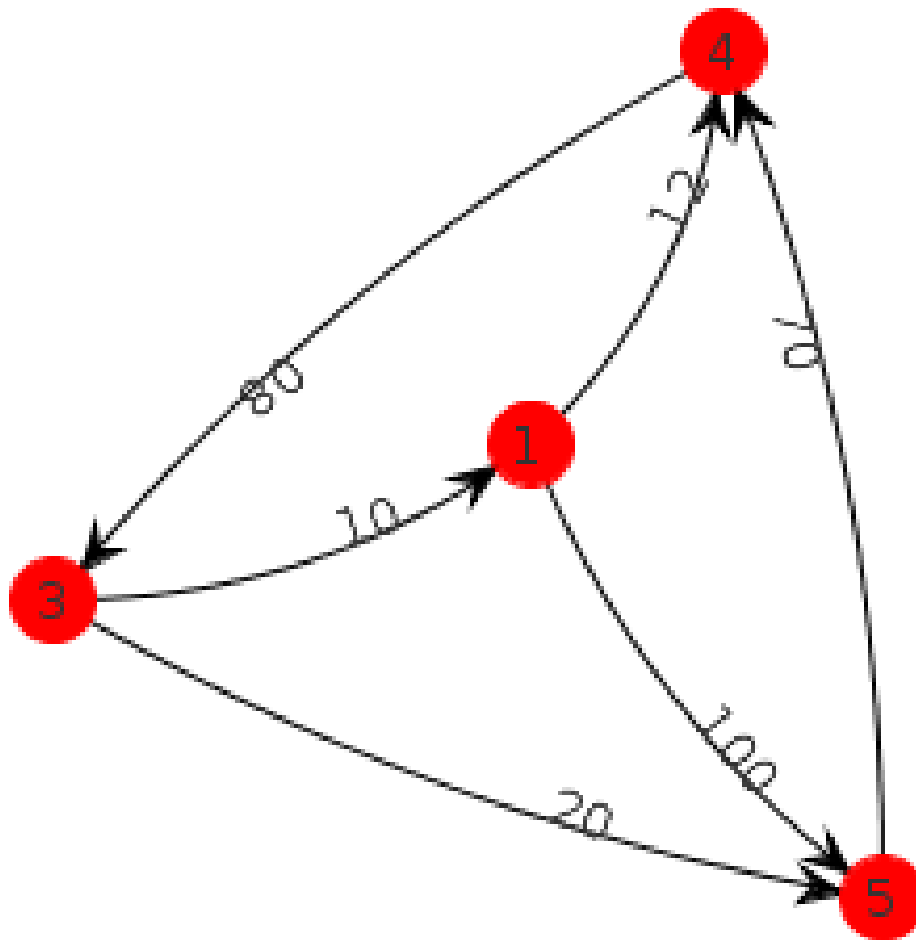


Fonte – Elaborado pelo autor.

grau de entrada diferente do grau de saída. Neste exemplo, o único vértice balanceado é o vértice 2. A Figura 54 mostra o grafo após a remoção deste vértice.

No próximo passo são calculadas as distâncias mínimas entre estes vértices utilizando o algoritmo de Dijkstra. O Quadro 5 mostra o cálculo destas distâncias. Lembrando que para que este passo possa ser realizado, o grafo deve ser fortemente conexo. Além disso, o cálculo da distância mínima entre os vértices de um grafo dirigido deve levar em consideração que a distância entre dois vértices i e j pode ser tal que, $dist(i, j) \neq dist(j, i)$, ou seja a distância mínima do vértice i para o vértice j pode ser diferente da distância mínima do vértice j para o

Figura 54 – Grafo para exemplo de execução do PCCD com vértices balanceados removidos



Fonte – Elaborado pelo autor.

vértice i .

Quadro 5 – Distâncias mínimas para o exemplo execução PCCD

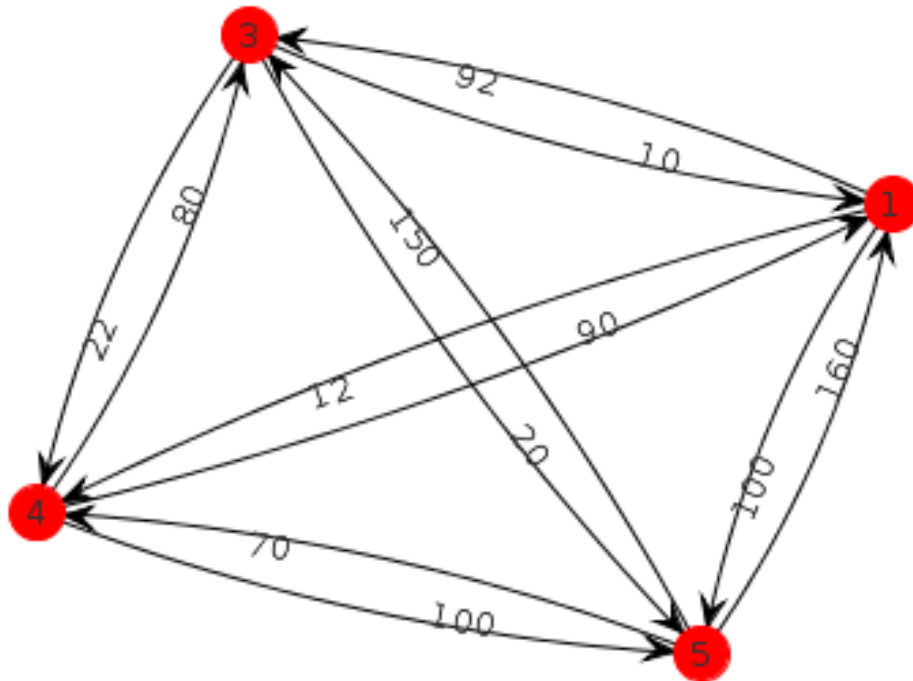
Vértice Origem	Vértice Destino	Distância Mínima	Caminho
1	3	192	1-4-3
1	4	12	1-4
1	5	100	1-5
3	1	10	3-1
3	4	22	3-1-4
3	5	20	3-5
4	1	90	4-3-1
4	3	80	4-3
4	5	100	4-3-5
5	1	160	5-4-3-1
5	3	150	5-4-3
5	4	70	5-4

Fonte – Elaborado pelo autor.

O próximo passo consiste em adicionar arcos no grafo, de tal maneira que cada arco

adicionado tenha o custo igual a distância mínima calculada no passo anterior. O grafo após as adições dos arcos é mostrado na Figura 55.

Figura 55 – Grafo para exemplo de execução do PCCD após adicionadas cópias de arcos



Fonte – Elaborado pelo autor.

No próximo passo do algoritmo o modelo para calcular o emparelhamento ótimo de um grafo direcionado deve ser gerado e executado. Para este exemplo, o modelo de PLI que foi gerado é o Modelo 5.1.

A primeira restrição deste modelo nos diz que o grau de entrada do vértice 5 é maior que o grau de saída em um, ou seja, $\deg^-(5) - \deg^+(5) = 1$, portanto para que este vértice se torne balanceado, ele deve duplicar algum dos arcos que saem dele (X51 ou X53).

$$\text{Minimizar } 160X_{51} + 100X_{45} + 90X_{41} + 20X_{35} + 12X_{14} + 100X_{15} + 80X_{43} + 22X_{34} + 10X_{31} + 150X_{53} + 92X_{13} + 70X_{54}$$

Sujeito a :

$$X_{51} + X_{53} = 1$$

$$X_{51} + X_{41} = 2$$

$$X_{43} + X_{53} = 1$$

$$X_{41} + X_{43} = 2$$

Na segunda restrição do modelo, o grau de saída do vértice 1 é maior que o grau de entrada em dois, ou seja, $\deg^+(1) - \deg^-(1) = 2$, portanto para que este vértice se torne balanceado, duas cópias de arcos que chegam nele devem ser adicionadas. Para a terceira restrição $\deg^+(3) - \deg^-(3) = 1$ e para a quarta restrição $\deg^-(4) - \deg^+(4) = 2$. A solução ótima para o emparelhamento neste grafo dirigido é mostrada no Quadro 6.

Quadro 6 – Resultado da execução do modelo de PLI 2 no exemplo de execução do PCCD

Variável	Valor
X15	0
X41	1
X14	0
X13	0
X45	0
X53	0
X43	1
X34	0
X51	1
X35	0
X31	0
X54	0

Fonte – Elaborado pelo autor.

A resposta do modelo de PLI nos informa que o grafo euleriano será obtido duplicando os arcos no caminho mínimo entre os vértices 4 e 1, entre os vértices 4 e 3 e entre os vértices 5 e 1. O grafo euleriano resultante deste procedimento é mostrado na Figura 56.

O último passo do algoritmo consiste em encontrar um circuito euleriano no grafo. A Figura 57 mostra o grafo em um passo intermediário do algoritmo de Fleury, onde os vértices e arcos destacados em amarelo foram os que já foram visitados. E a Figura 58 mostra o grafo no final da execução, com todos os vértices e arcos visitados.

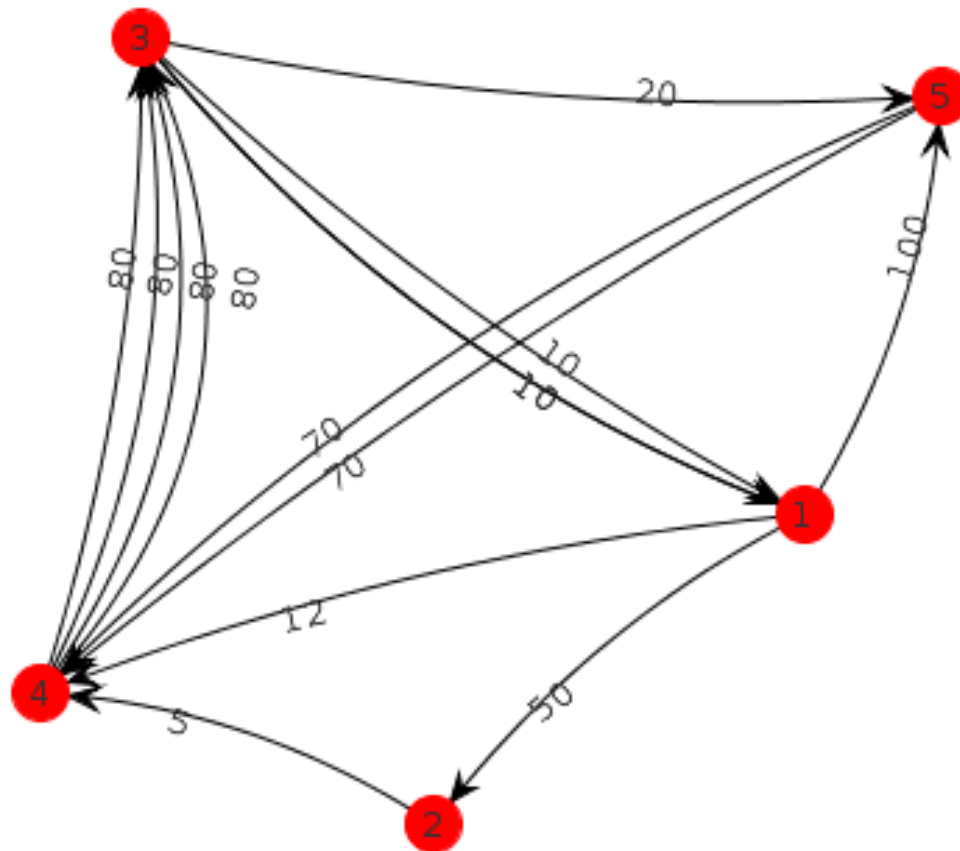
O Quadro 7 mostra algumas informações desta execução do PCCD.

Quadro 7 – Informações da execução do exemplo de PCCD

Soma dos custos das arestas duplicadas	330
Número de arestas duplicadas	6
Arestas duplicadas	(4,3) , (3,1) e (5,4)
Soma do percurso do carteiro	677
Circuito euleriano	1-4-3-1-5-4-3-1-2-4-3-5-4-3-1

Fonte – Elaborado pelo autor.

Figura 56 – Grafo euleriano para exemplo de execução do PCCD após adicionadas cópias de arcos



Fonte – Elaborado pelo autor.

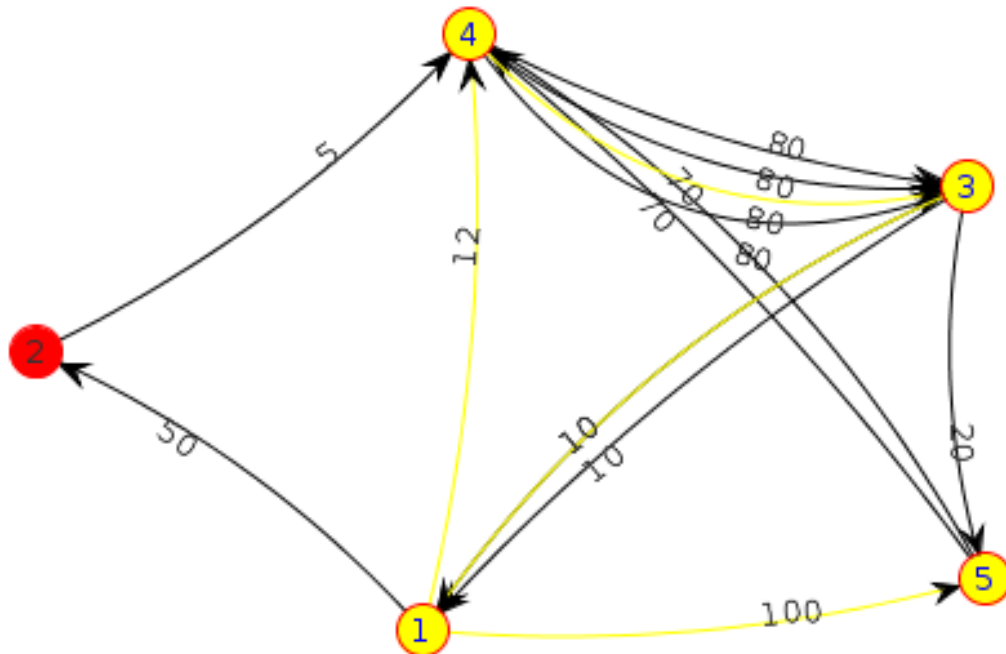
5.3 PCC MISTO

O algoritmo que resolve o PCC misto é exatamente o mesmo algoritmo do PCCD. Claro, no entanto, que o grafo misto deve ser transformado em um grafo dirigido antes do início do algoritmo. O Fluxograma da execução do algoritmo é quase o mesmo apresentado na Figura 51, foi adicionado somente o passo de transformar o grafo misto em um grafo dirigido equivalente. A Figura 59 mostra o novo fluxograma.

O Algoritmo 13 descreve o algoritmo que encontra um circuito euleriano em um grafo misto. As funções `CALCULAR_EMPARELHAMENTO_PERFEITO`, `TORNAR_GRAFO_EULERIANO` e `ENCONTRAR_CIRCUITO_EULERIANO` são idênticas as descritas nas seções anteriores. Inclusive o Modelo de PLI 2 é o modelo matemático utilizado para calcular o emparelhamento perfeito.

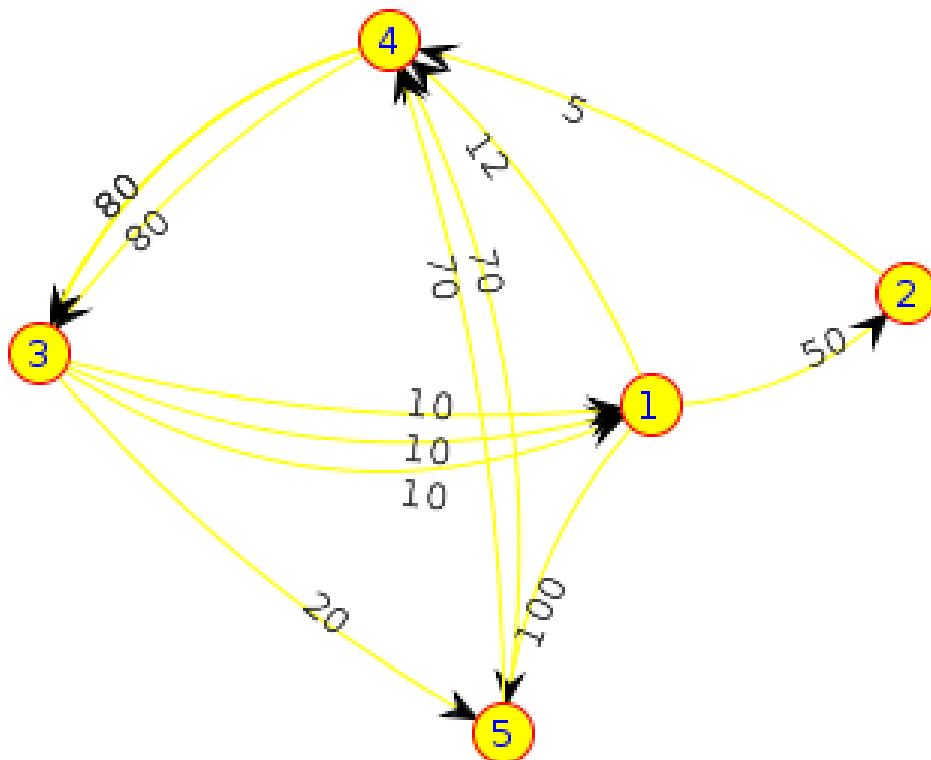
O Algoritmo 15 descreve como um grafo misto é transformado em um grafo dirigido. A ideia do algoritmo é substituir cada aresta do grafo por dois arcos opostamente dirigidos e com

Figura 57 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCD



Fonte – Elaborado pelo autor.

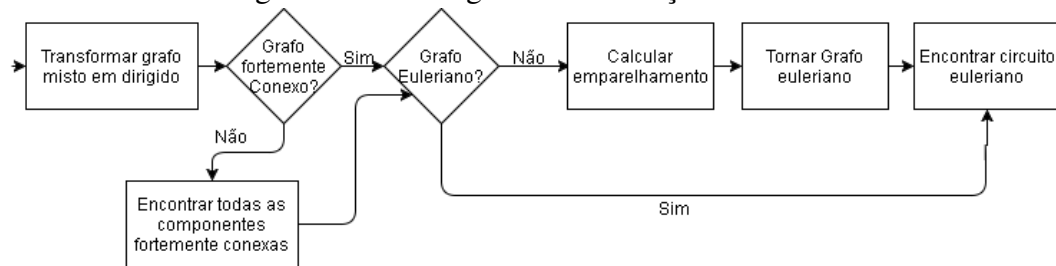
Figura 58 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCD



Fonte – Elaborado pelo autor.

o mesmo custo da aresta original, ou seja, para toda aresta e_{ij} com custo d_{ij} criar dois arcos a_{ij} e a_{ji} , ambos com custo d_e .

Figura 59 – Fluxograma da execução do PCCM



Fonte – Elaborado pelo autor.

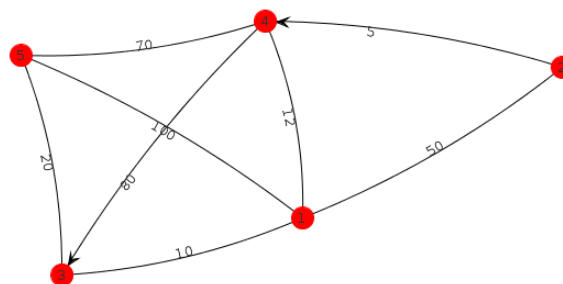
Se um grafo misto G for inicialmente não euleriano, um circuito euleriano pode ser encontrado seguindo os passos:

- Transformar o grafo G misto em um grafo dirigido equivalente, utilizando o Algoritmo 15;
- Gerar um grafo G' que é uma cópia de G porém remover os vértices de grau par. O grafo G' terá m vértices;
- Calcular as menores distâncias entre estes m vértices em G utilizando o Algoritmo 6 de Dijkstra (??), descrito na Seção 3.5.1;
- Adicionar arestas em G' , de tal maneira que cada aresta adicionada tenha o custo igual a distância mínima calculada no item anterior. Após este procedimento obtém-se um grafo completo K_m ;
- Encontrar o emparelhamento ótimo entre estes vértices utilizando o Modelo de PLI 2;
- Adicionar estes $m/2$ caminhos mínimos, como arestas ligando os vértices do emparelhamento no grafo G . G após este passo é euleriano.
- Encontrar um circuito euleriano em G .

5.3.1 Execução do Algoritmo para o PCCM

O grafo como exemplo de execução do PCCM é mostrado na Figura 60.

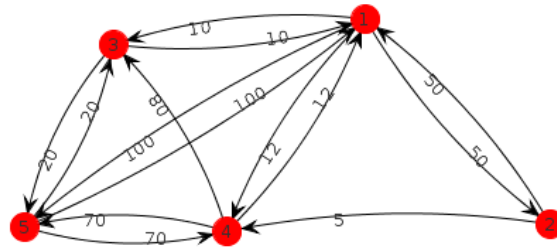
Figura 60 – Grafo para exemplo de execução do PCCM



Fonte – Elaborado pelo autor.

O primeiro passo do algoritmo que calcula o PCCM é transformar o grafo misto em um grafo dirigido equivalente. O grafo resultante desta transformação é mostrado na Figura 61.

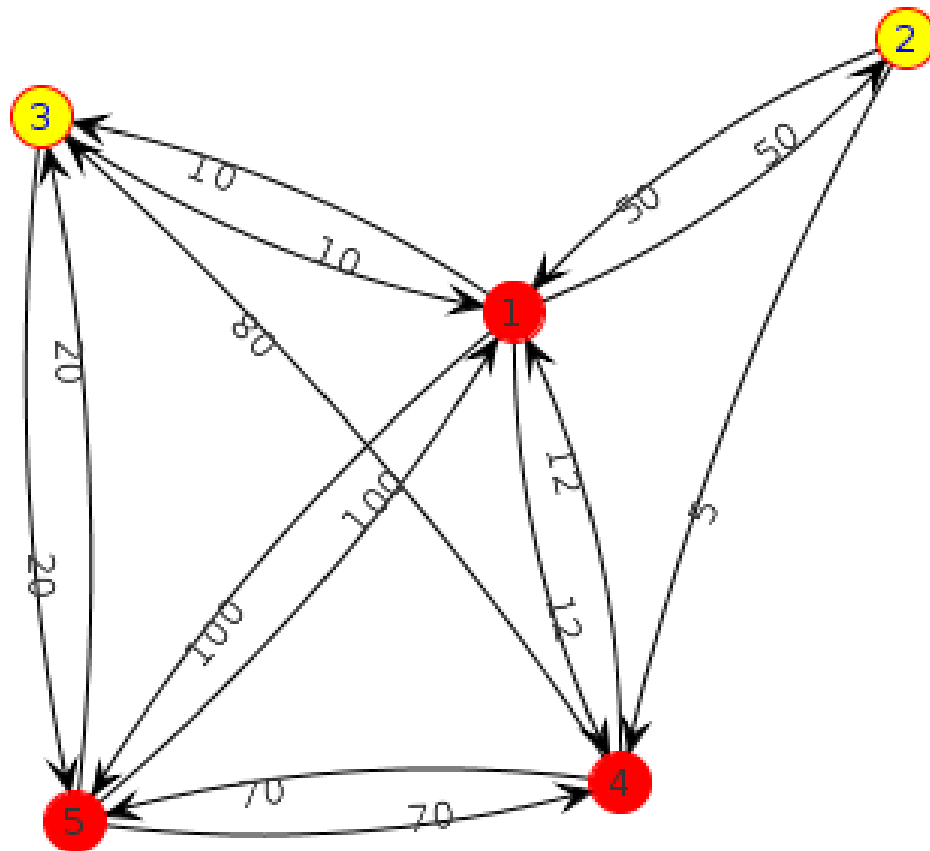
Figura 61 – Grafo para exemplo de execução do PCCM após transformação em grafo dirigido



Fonte – Elaborado pelo autor.

Observa-se que no grafo existem dois vértices desbalanceados, o vértice 2 e o vértice 3. A Figura 62 mostra este grafo com os vértices desbalanceados destacados.

Figura 62 – Grafo para exemplo de execução do PCCM com vértices desbalanceados destacados



Fonte – Elaborado pelo autor.

O próximo passo da execução do algoritmo consiste em remover todos os vértices balanceados do grafo, pois, assim como no algoritmo do PCCD, o emparelhamento é realizado

somente entre os vértices desbalanceados. A Figura 63 mostra como fica o grafo resultante desta remoção. Como somente os vértices 2 e 3 estão desbalanceados e como não existe arcos ligando o vértice 2 ao 3 e nem ligando o vértice 3 ao 2, o grafo resultante desta remoção é bem simples.

Figura 63 – Grafo para exemplo de execução do PCCM com vértices balanceados removidos



Fonte – Elaborado pelo autor.

No próximo passo são calculadas as distâncias mínimas entre estes vértices utilizando o algoritmo de Dijkstra. O Quadro 8 mostra o cálculo destas distâncias.

Quadro 8 – Distâncias mínimas para o exemplo execução PCCM

Vértice Origem	Vértice Destino	Distância Mínima	Caminho
2	3	27	2-4-1-3
3	2	60	3-1-2

Fonte – Elaborado pelo autor.

O próximo passo consiste em adicionar arcos neste grafo, de tal maneira que cada arco adicionado tenha o custo igual a distância mínima calculada no passo anterior. O grafo após as adições dos arcos é mostrado na Figura 64.

No próximo passo do algoritmo o modelo de PLI para calcular o emparelhamento ótimo de um grafo dirigido deve ser gerado e executado, bem como foi realizado no algoritmo

Figura 64 – Grafo para exemplo de execução do PCCM após adicionadas cópias de arcos



Fonte – Elaborado pelo autor.

do PCCD. Para este exemplo, o modelo que foi gerado é o Modelo 5.1.

$$\text{Minimizar } 27X_{23} + 60X_{32}$$

Sujeito a :

$$X_{32} = 1$$

$$X_{32} = 1$$

As restrições deste modelo são iguais e significam que o grau de saída do vértice 2 é maior que o grau de entrada em 1, ou seja, $\deg^+(2) - \deg^-(2) = 1$ e dizem também que o grau de entrada do vértice 3 é maior que o grau de saída dele em 1, portanto $\deg^-(3) - \deg^+(3) = 1$. Visto isso, para balancear ambos os vértices basta adicionar uma cópia de aresta ligando o vértice 3 ao vértice 2, e é exatamente isto que o resultado do modelo de PLI mostrado no Quadro 9 nos informa.

Quadro 9 – Execução do modelo de PLI 2 no exemplo do PCCM

Variável que foi selecionada	Valor da Variável
X32	1

Fonte – Elaborado pelo autor.

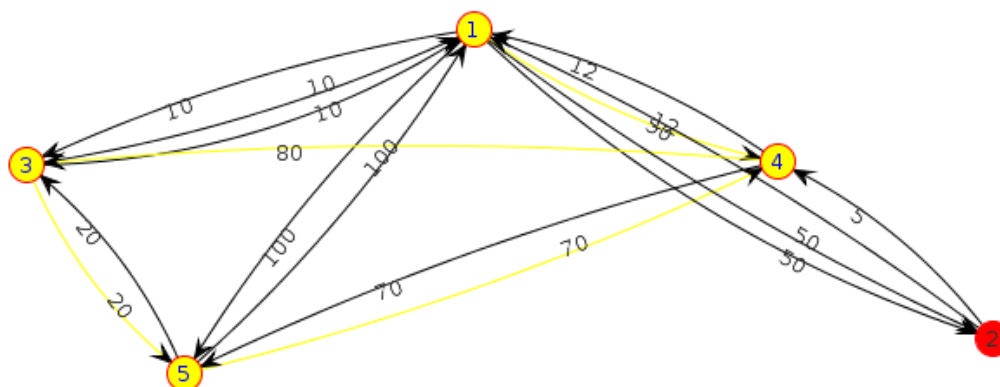
O último passo do algoritmo consiste em encontrar um circuito euleriano no grafo . A Figura 65 mostra o grafo em um passo intermediário do algoritmo de Fleury, onde os vértices e arcos destacados em amarelo foram os que já foram visitados.

A Figura 66 mostra o grafo no final da execução, com todos os vértices e arcos visitados.

O Quadro 10 mostra algumas informações desta execução do PCCM.

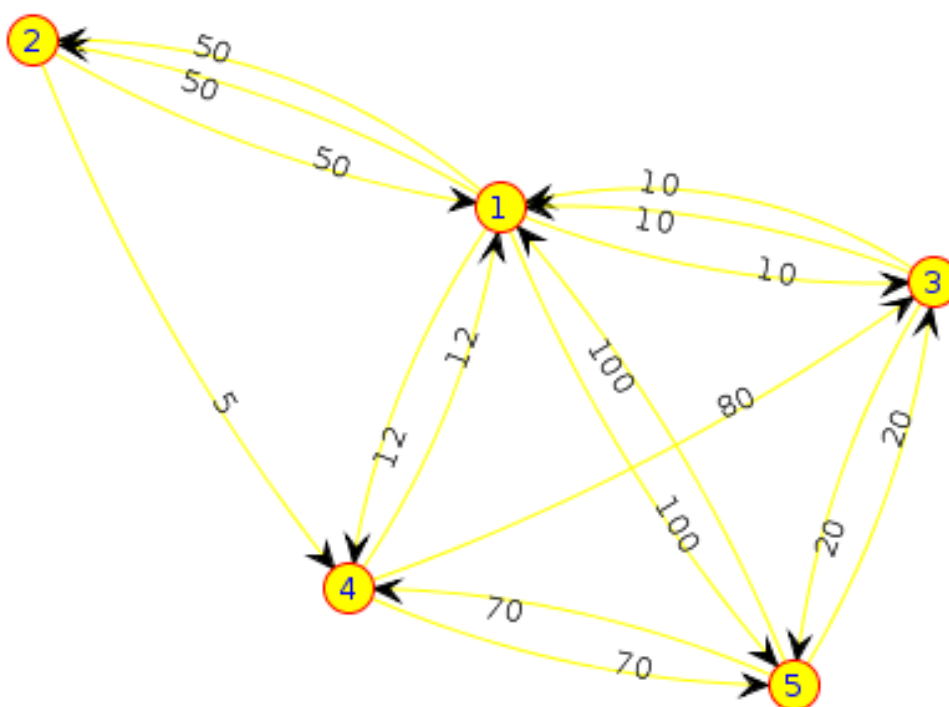
Os modelos de PLI utilizados nestes algoritmos são calculados utilizando o Solver CPLEX (????). A próxima seção é destinada a mostrar o funcionamento do CPLEX bem como

Figura 65 – Construção intermediária de circuito euleriano no grafo do exemplo de execução do PCCM



Fonte – Elaborado pelo autor.

Figura 66 – Construção total de circuito euleriano no grafo do exemplo de execução do PCCM



Fonte – Elaborado pelo autor.

Quadro 10 – Informações da execução do exemplo de PCCM

Soma dos custos das arestas duplicadas	60
Número de arestas duplicadas	1
Arestas duplicadas	(3,2)
Soma do percurso do carteiro	669
Circuito euleriano	1-2-4-5-3-1-2-1-3-5-4-3-1-5-1-4-1

Fonte – Elaborado pelo autor.

descrever como preparar os modelos para a execução nele, mais precisamente será descrito o algoritmo GERAR_MODELAGEM_MATEMATICA.

5.4 GERAÇÃO MODELAGEM MATEMÁTICA UTILIZANDO CPLEX

No início desta seção será apresentado as principais interfaces utilizadas do CPLEX para criar os modelos. A principal interface do CPLEX é a **IloCplex**, que implementa as interfaces **IloModeler**, **IloMPModeler** e **IloCplexModeler**. Estas interfaces definem construtores para modelar objetos dos tipos mostrados na Tabela 1 e que podem ser usados por um objeto **IloCplex** para manusear um modelo matemático.

Tabela 1 – Principais objetos CPLEX

Interface CPLEX	Tipos de modelagem
IloNumVar	Modelagem de variáveis do modelo matemático
IloRange	Modelagem de restrições do modelo matemático
IloObjective	Modelagem da função objetivo do modelo matemático
IloLinearNumExpr	Modelagem de expressões utilizando variáveis IloNumVar

Fonte – Elaborado pelo autor.

A Tabela 2 mostra exemplos dos principais objetos e operações mais utilizadas do CPLEX.

Tabela 2 – Exemplos das principais operações do CPLEX

IloCplex cplex = new IloCplex()	Cria uma variável cplex para manusear o modelo matemático
IloLinearNumExpr fo = cplex.linearNumExpr()	Cria a função objetivo do modelo matemático
cplex.add(fo)	Adiciona a função objetivo ao modelo matemático
IloNumVar variavel = cplex.boolVar("X1")	Cria a variável X1
cplex.prod(2.0, x[1])	Realiza o produto $2 \cdot x[1]$
cplex.sum(x[0], cplex.prod(3.0, x[2]))	Realiza a soma $x[0] + 3 \cdot x[2]$
IloLinearNumExpr expr = cplex.linearNumExpr()	Cria uma expressão matemática
expr.addTerm(2.0, x[0])	Adiciona $2 \cdot x[0]$ na expressão expr
cplex.solve()	Resolve o modelo matemático

Fonte – Elaborado pelo autor

O Algoritmo 16 mostra os passos para se criar um modelo de PLI para calcular o emparelhamento perfeito a partir de um grafo completo. O algoritmo é mostrado para um grafo completo pois para todos os tipos de PCC calculados neste trabalho um grafo completo é gerado antes da criação do modelo de PLI que calcula o emparelhamento perfeito. Este algoritmo foi modularizado em diferentes algoritmos para facilitar o seu entendimento. As funções nativas da biblioteca CPLEX são demarcadas em negrito nos algoritmos a seguir.

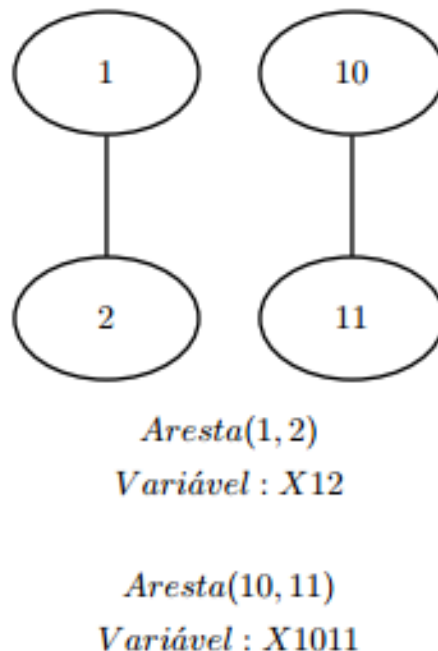
Os algoritmos **ADICIONA_VARIAVEIS_CPLEX**, **ADICIONA_FUNCAO_OBJETIVO** e **ADICIONA_TIPO_FUNCAO_OBJETIVO** são iguais para os três tipos de PCC e serão descri-

tas a seguir. O algoritmo ADICIONA_RESTRICOES_CPLEX difere de acordo com o tipo de PCC estudado, por isso, este será descrito para cada tipo de PCC em uma seção distinta.

O algoritmo ADICIONA_VARIAVEIS_CPLEX (Algoritmo 17) gera, a partir de um grafo completo, um conjunto de variáveis que comporão o modelo de PLI para o cálculo do emparelhamento. Nesta modelagem todas as arestas do grafo serão transformadas em uma variável no modelo.

O procedimento TRANSFORMA_ARESTA_VARIAVEL (Algoritmo 18) converte uma aresta do grafo em uma variável para ser adicionada ao modelo. Os nomes das variáveis foram padronizados como é mostrado na Figura 67.

Figura 67 – Padronização dos nomes das variáveis do modelo de PLI criado

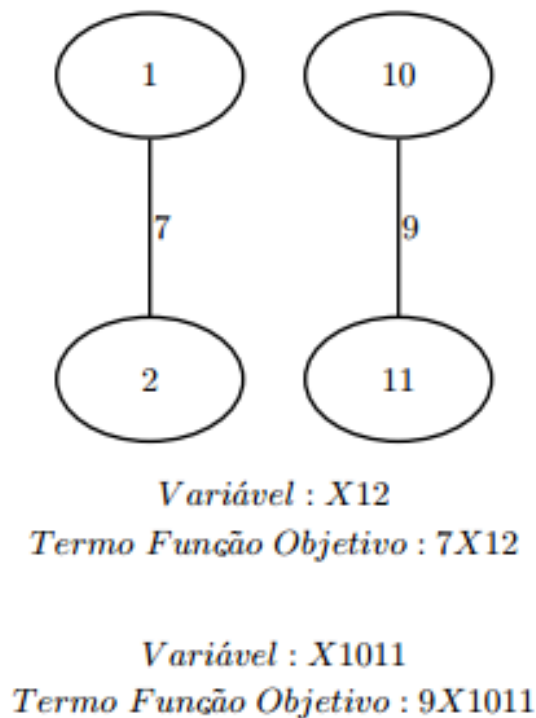


Fonte – Elaborado pelo autor.

O algoritmo ADICIONA_FUNCAO_OBJETIVO (Algoritmo 19) adiciona as variáveis do modelo de PLI à função objetivo. Cada termo da função objetivo é uma variável precedida do custo de percorrer a aresta equivalente, como demonstrado na Figura 68.

O algoritmo ADICIONA_TIPO_FUNCAO_OBJETIVO (Algoritmo 20) adiciona qual o tipo de modelo de PLI que será criado. Todos os modelos criados serão modelos de minimização, pois no PCC deseja-se encontrar o emparelhamento perfeito.

Figura 68 – Montagem de cada termo do modelo de PLI criado



Fonte – Elaborado pelo autor.

5.4.1 Adição das restrições aos modelos de PLI utilizando CPLEX

Como mencionado anteriormente, cada tipo de PCC tem características distintas, por isso cada um têm a construção das restrições do modelo de PLI realizada de maneira distinta. As seções seguintes especificam esta construção para cada tipo de PCC estudado.

5.4.1.1 PCC não dirigido

O algoritmo ADICIONA_RESTRICICOES_CPLEX para o caso do PCC não dirigido (Algoritmo 21) é mostrado abaixo.

5.4.1.2 PCC dirigido

O algoritmo para adicionar as restrições do modelo de PLI para o PCC dirigido é o Algoritmo 22.

O grande diferencial das restrições deste modelo para as restrições do modelo não dirigido é que neste modelo a soma de cada restrição é forçada a ser igual à diferença entre os graus de entrada e saída de cada vértice. Portanto, se o grau de entrada do vértice for maior que

o grau de saída dele, a restrição será igualada a $\deg^- - \deg^+$, caso contrário, ou seja, se o grau de saída do vértice for maior que seu grau de entrada, a restrição será igualada a $\deg^+ - \deg^-$.

5.4.1.3 PCC misto

Como neste trabalho o PCC misto é resolvido transformando um grafo misto em um grafo direcionado equivalente o modelo de PLI é gerado de maneira similar em ambos os casos, desta maneira o algoritmo ADICIONA_RESTRICOES_CPLEX é idêntico ao Algoritmo 22 da subseção anterior.

6 RESULTADOS

6.1 EXEMPLO DE COLETA DE LIXO NA CIDADE DE FORTALEZA

Segundo Laporte (??), o problema de roteirização de veículos consiste em definir roteiros de veículos que minimizem o custo total de atendimento, cada um dos quais iniciando e terminando no depósito dos veículos, assegurando que cada ponto seja visitado exatamente uma vez e a demanda em qualquer rota não exceda a capacidade do veículo que a atende. Descobrir qual rota um caminhão de lixo deve percorrer para minimizar sua distância percorrida é um problema de roteirização de veículos e é uma aplicação recorrente (????) do problema do carteiro chinês.

Nesta seção será fornecido um exemplo aplicado à coleta de lixo em algumas ruas selecionadas do bairro Monte Castelo da cidade de Fortaleza. O caminhão de lixo deve passar pelos pontos de coleta de lixo que estão situados nas ruas, podendo assim haver mais de um ponto de coleta em uma determinada rua. As ruas passadas pelo caminhão são mostradas na tabela 3.

A figura 69 mostra a disposição destas ruas.

Ao executar o algoritmo proposto neste texto foram obtidos os resultados mostrados na tabela 4.

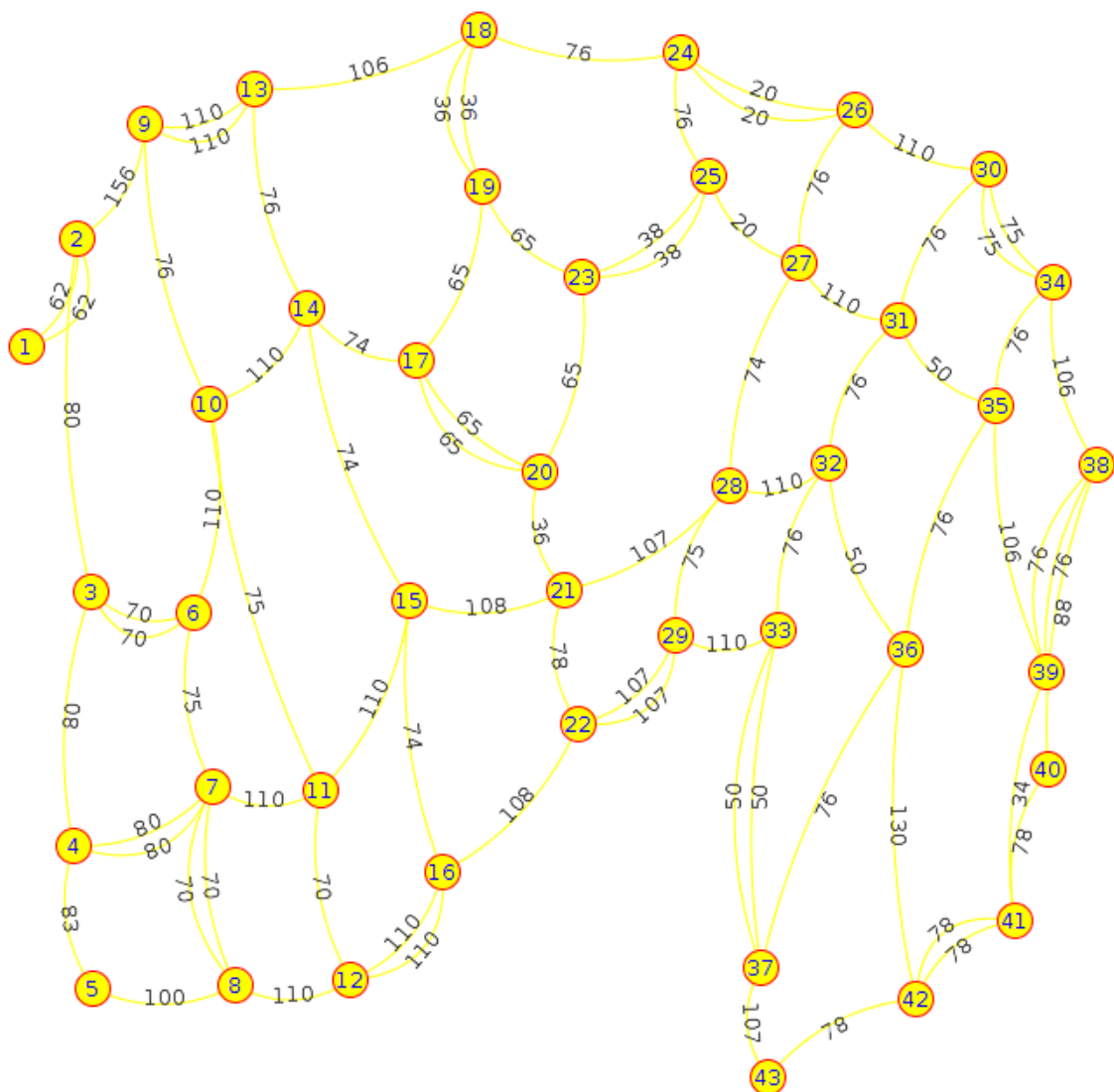
O grafo após a animação do percurso do caminhão é o da figura 70. Todos os vértices e todas as arestas estão coloridos pois todos foram visitados durante o percurso do caminhão de lixo.

Tabela 3 – Ruas da cidade de Fortaleza a serem percorridas

1	Rua Padre Frota
2	Rua Conrado Cabral
3	Rua Benjamin Barroso
4	Rua José Cândido
5	Avenida Sargento Hermínio
6	Rua Aníbal Mascarenhas
7	Rua Demócrito Rocha
8	Rua Henrique Autran
9	Rua Antônio Dumont
10	Rua Leiria de Andrade
11	Rua Casimiro Montenegro
12	Rua José Marrocos
13	Rua Soares Bulcão
14	Avenida Bezerra de Menezes

Fonte – Elaborado pelo autor

Figura 70 – Representação do problema de coleta de lixo como um grafo



Fonte – Elaborado pelo autor.

Tabela 4 – Resultados obtidos na execução do exemplo

Tempo de Execução (s)	1
Soma das arestas duplicadas	1047
Número de arestas duplicadas	15
Soma de todas as arestas	6595
Percurso do caminhão	1,2,9,10,6,7,11,15,14,10,11,12,16,22, 29,28,27,25,24,18,13,9,13,14,17, 20,17,19,18,19,23,25,23,20,21,22,29,33,37,36,35,39,38,39,41, 42,36,32,33,37,43,42,41,40,38,34,30,34,35,31,27,26,24,26,30,31, 32,28,21,15,16,12,8,5,4,7,8,7,4,3,6,3,2,1

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho se propõe a codificar uma solução para os três principais tipos de PCC: o dirigido, o não dirigido e o misto. Além disso este trabalho tem como objetivo mostrar de uma maneira gráfica e amigável o circuito do carteiro enquanto passa pelos vértices e arestas do grafo. Estes objetivos foram alcançados pelo presente trabalho, pois foi apresentado uma maneira exata de calcular cada um destes tipos bem como uma animação do circuito do carteiro pelo grafo estudado.

7.1 CONTRIBUIÇÕES DO TRABALHO

Este trabalho tem como contribuições:

- Fornece uma maneira exata de calcular o PCC dirigido, o não dirigido e o misto
- Fornece um ambiente interativo de visualização do percurso do carteiro

7.2 LIMITAÇÕES

Como este trabalho utiliza o CPLEX como solver, este, em sua versão gratuita, suporta modelos matemáticos de até 1000 variáveis. Por este motivo se o modelo matemático gerado por este programa tiver mais que 1000 variáveis este não irá funcionar. Havendo uma licença que não possua limite de variáveis este trabalho pode resolver qualquer instância de problema.

7.3 TRABALHOS FUTUROS

Os trabalhos futuros que virão a partir deste trabalho podem dar enfoque especial em melhorar o ambiente de visualização do carteiro, se aproveitando do que foi feito neste trabalho.

Outros trabalhos futuros podem partir para tentar solucionar problemas de porte bem maior, com quantidade de vértices e arestas bem maiores que a deste trabalho. Para tais trabalhos o enfoque pode ser modificado para utilizar meta-heurísticas para obter um menor tempo, ou ainda buscar uma solução mista que envolva programação linear inteira e meta-heurísticas.

APÊNDICE

APÊNDICE A – ALGORITMOS

Algoritmo 1: Algoritmo Blossom I

Entrada: $G = (V, E)$

Matching M em G

Saída: Matching máximo M^* em G

ENCONTRAR_MATCHING_MAXIMO(G, M)

início

$P = \text{ENCONTRAR_CAMINHO_AUMENTADO}(G, M)$

if $P \neq \emptyset$ **then**

return $\text{ENCONTRAR_MATCHING_MAXIMO}(G, \text{aumento de } M \text{ sobre } P)$

end

if $P = \emptyset$ **then**

return M

end

fim

Algoritmo 2: Algoritmo que adiciona o tipo de função objetivo

Entrada: `IloCplex M`, `IloLinearNumExpr funcao_objetivo`

início

$M.\text{addMinimize}(\text{funcao_objetivo})$ // Adiciona tipo de função de
 minimização

fim

Algoritmo 3: Algoritmo de Hierholzer

Entrada: $G' = G$ com $G' = (V', E')$
 $n_1 \in G'$ (vértice inicial)

Saída: C contendo o circuito euleriano

início
 $C = [n_1]$ // Inicialmente somente com o vértice inicial

while $E \neq \emptyset$ **do**
 $n_i =$ um vértice de C tal que $d(n_i) > 0$ em G' // Vértice adjacente

 $C' =$ Circuito em G' que contém n_i
 $G' = G' - a$ // Retira aresta $a \in E'$ do grafo

Em C , substituir o vértice n_i pelo circuito C'
end

Retorne C
fim

Algoritmo 4: Algoritmo de Fleury

Entrada: $G' = G$ com $G' = (V', E')$
 $n_1 \in G'$ (vértice inicial)

Saída: C contendo o circuito euleriano

início
 $C = [n_1]$ // Inicialmente somente com o vértice inicial

 $v = n_1$ // Vértice Inicial

while $E \neq \emptyset$ **do**

Escolha uma aresta $a(v, w)$ adjacente de v desde que ela não seja uma ponte

 $v = w$ // Atribuindo vértice para próxima iteração

 $C = C \cup v$ // Colocando o próximo vértice no circuito euleriano

 $G' = G' - a$ // Retira aresta $a \in E'$ do grafo

end

Retorne C
fim

Algoritmo 5: Algoritmo Blossom I

Entrada: $G = (V, E)$

Matching M em G

Saída: Matching máximo M^* em G

ENCONTRAR_MATCHING_MAXIMO(G, M)

início

$P = \text{ENCONTRAR_CAMINHO_AUMENTADO}(G, M)$

if $P \neq \emptyset$ **then**

 | **return** $\text{ENCONTRAR_MATCHING_MAXIMO}(G, \text{aumento de } M \text{ sobre } P)$

end

if $P = \emptyset$ **then**

 | **return** M

end

fim

Algoritmo 6: Algoritmo de Dijkstra

Entrada: $G(N, E)$

$n_{\text{inicial}} \in N$

Saída: Distâncias de n_{inicial} para todos os outros vértices de G

início

$\text{INICIALIZA}(G, n_{\text{inicial}})$

$\text{VISITADOS} = \emptyset$ // Conjunto de vértices visitados é vazio no início

$Q_{\text{prioridade}} = n_{\text{inicial}}$ // Fila de prioridade inicializada com o vértice inicial

while $Q_{\text{prioridade}} \neq \emptyset$ **do**

$u = Q_{\text{prioridade}}.\text{EXTRAI_MINIMO}$ // Extrai o primeiro vértice da Fila

$\text{VISITADOS} = \text{VISITADOS} \cup u$ // Adiciona u ao conjunto dos vértices visitados

for v tal que v está na vizinhança de u **do**

 | $\text{RELAXA}(u, v)$

end

end

return G

fim

Algoritmo 7: Algoritmo de Relaxação de Dijkstra

RELAXA(u, v)**início**

// Desigualdade triangular

if $\delta_v \geq \delta_u + d(u, v)$ **then** | $\delta_v = \delta_u + d(u, v)$ **end****fim**

Algoritmo 8: Inicialização do Algoritmo de Dijkstra

INICIALIZA($G, n_{inicial}$)**início** **for** $n_i \in N$ **do** | $\delta_{n_i} = \infty$ **end** $\delta_{n_{inicial}} = 0$ // Vértice inicial tem custo zero para ele mesmo**fim**

Algoritmo 9: Algoritmo de Bellman-Ford

Entrada: $G(N, E)$ $n_{inicial} \in N$ **Saída:** Distâncias de $n_{inicial}$ para todos os outros vértices de G ou FALSO se houver ciclos negativos**início** INICIALIZA($G, n_{inicial}$) // $|N[G]|$ é a quantidade de vértices de G **for** $i = 1$ até $|N[G]| - 1$ **do** // $E[G]$ é o conjunto das arestas de G **for** Cada aresta $(u, v) \in E[G]$ **do** RELAXA(u, v) **end** **end** **if** CHECA_CICLOS_NEGATIVOS(G) = FALSO **then** **return** FALSO // G possui ciclos negativos, por isso não existe
 solução apropriada **end** **return** G **fim**

Algoritmo 10: Algoritmo Básico resolução do PCC

Entrada: Grafo G

Saída: Circuito do carteiro C , se existir

início

```

if  $G$  for desconexo then
  // Sem solução para grafos desconexos
  SAIR
end
if  $G$  for não euleriano then
  CALCULAR_EMPARELHAMENTO_PERFEITO
  TORNAR_GRAFO_EULERIANO
   $C$  = ENCONTRAR_CIRCUITO_EULERIANO
end
if  $G$  for euleriano then
  |  $C$  = ENCONTRAR_CIRCUITO_EULERIANO
end
return  $C$ 

```

fim

Algoritmo 11: Algoritmo para calcular o emparelhamento ótimo do grafo

Entrada: Grafo G

Saída: Conjunto de Arestas A do emparelhamento

início

```

  Adicionar os  $m$  vértices de grau ímpar em  $G'$ 
  Encontrar as distâncias mínimas entre todos estes  $m$  vértices de  $G'$ 
  Adicionar arestas em  $G'$  com estas distâncias calculadas de maneira que  $G'$  se torne um
  grafo completo  $K_m$ 
   $Model$  = GERAR_MODELAGEM_MATEMATICA( $G'_m$ )
   $A$  = EXECUTAR_MODELO( $Model$ )
  return  $A$ 

```

fim

Algoritmo 12: Algoritmo para tornar um grafo euleriano

Entrada: Conjunto de arestas E do emparelhamento**Saída:** Grafo G euleriano**início** **while** *existirem arestas e em E ainda não copiadas em G* **do** | Adicionar cópia de e em G **end** **return** G **fim**

Algoritmo 13: Algoritmo para resolução do PCCM

Entrada: Grafo G **Saída:** Circuito do carteiro C , se existir**início** **if** G *for desconexo* **then**

| // Sem solução para grafos desconexos

| SAIR

end

TRANSFORMAR_GRAFO_EM_DIRIGIDO

if G *for não euleriano* **then**

| CALCULAR_EMPARELHAMENTO_PERFEITO

| TORNAR_GRAFO_EULERIANO

 | $C = \text{ENCONTRAR_CIRCUITO_EULERIANO}$ **end** **if** G *for euleriano* **then** | $C = \text{ENCONTRAR_CIRCUITO_EULERIANO}$ **end** **return** C **fim**

Algoritmo 14: Checagem de ciclos negativos no Algoritmo de Bellman-Ford

CHECA_CICLOS_NEGATIVOS(G)

início

for *Cada aresta* $(u, v) \in E[G]$ **do**

if $\delta_v \geq \delta_u + d(u, v)$ **then**

return *FALSO*

end

end

return *VERDADEIRO*

fim

Algoritmo 15: Algoritmo para transformar um grafo misto em um grafo dirigido equivalente

Entrada: Grafo misto $G(N, A, E)$

Saída: Grafo dirigido $G(N, A)$

início

while *existirem arestas* $e \in E$ *em* G **do**

 Selecionar e_{ij} de E

 Adicionar arcos a_{ij} e a_{ji} em A com custo d_e em G

 Remover e de E

end

return G

fim

Algoritmo 16: Algoritmo para gerar o modelo de PLI utilizando o solver CPLEX

Entrada: Grafo completo G' com m arestas

Saída: Modelo de PLI M gerado

início

```

IloCplex  $M$  = IloCplex() // Variavel para manusear o modelo matemático
    no CPLEX
IloNumVar[] variaveis = IloNumVar[ $m$ ] // Vetor de Variáveis
IloLinearNumExpr funcao_objetivo =  $M$ .linearNumExpr() // Variável com a
    função objetivo
List<IloRange> restricoes = ArrayList() // Restrições do modelo
 $M$ .VARS = ADICIONA_VARIAVEIS_CPLEX( $G'$ , variaveis)
 $M$ .FUN_OBJ = ADICIONA_FUNCAO_OBJETIVO( $G'$ , funcao_objetivo, variaveis)
 $M$ .TIPO_FO = ADICIONA_TIPO_FUNCAO_OBJETIVO(funcao_objetivo)
 $M$ .REST = ADICIONA_RESTRICOES_CPLEX( $G'$ ,  $M$ , funcao_objetivo, restricoes,
    variaveis)
return  $M$  // Retorna o modelo de PLI com suas variáveis e restrições

```

fim

Algoritmo 17: Algoritmo para adicionar as variáveis ao modelo de PLI

Entrada: Grafo G' , **IloCplex** M , Vetor variaveis vazio

Saída: Vetor com as variáveis do modelo

início

```

i = 0 // Contador para o vetor de variáveis
while existirem arestas e ainda não visitadas em  $G'$  do
    variavel = TRANSFORMA_ARESTA_VARIAVEL( $e$ ) // Transforma a
        aresta em uma variável para o modelo matemático
    variaveis[i] =  $M$ .boolVar(variavel) // Adiciona a variavel ao modelo
        matemático
    i++
end
return variaveis

```

fim

Algoritmo 18: Algoritmo que converte uma aresta em uma variável a ser adicionada ao modelo de PLI

Entrada: Aresta e

Saída: Variável var

início

```

    v1 = e.getVertice1() // Retira vértice 1 da aresta e
    v2 = e.getVertice2() // Retira vértice 2 da aresta e
    var = "X" + v1 + v2 // Forma o nome da variável

```

fim

Algoritmo 19: Algoritmo que adiciona cada termo à função objetivo do modelo de PLI

Entrada: Grafo G' , **IloLinearNumExpr** $funcao_objetivo$, **IloNumVar[]** $variaveis$

Saída: **IloLinearNumExpr** $funcao_objetivo$

início

```

    i = 0 // Contador para o vetor de variáveis
    while existem arestas e ainda não visitadas em  $G'$  do
        funcao_objetivo.addTerm(e.getCusto(), variaveis[i]) // Adicionada cada
            termo à função objetivo do modelo de PLI
        i++
    end
    return funcao_objetivo

```

fim

Algoritmo 20: Algoritmo que adiciona o tipo de função objetivo

Entrada: **IloCplex** M , **IloLinearNumExpr** $funcao_objetivo$

início

```

    M.addMinimize(funcao_objetivo) // Adiciona tipo de função de
        minimização

```

fim

Algoritmo 21: Algoritmo que adiciona as restrições à modelagem matemático do PCC não dirigido

Entrada: Grafo G' , **IloCplex** M , **IloLinearNumExpr** $funcao_objetivo$, $List<IloRange>$ $restricoes$, **IloNumVar**[] $variaveis$

início

```

IloLinearNumExpr ils // Variavel CPLEX para manipulação da expressão
numérica
IloNumVar variavel // Variavel CPLEX para uso no modelo de PLI
while existirem vértices  $n$  ainda não visitadas em  $G'$  do
    ils =  $M.linearNumExpr()$ 
    while existirem vértices adjacentes  $nAdj$  de  $n$  não visitadas do
        variavel =  $TRANSFORMA\_ARESTA\_VARIABEL(n, nAdj)$  // Transforma
        a aresta  $(n, nAdj)$  em uma variável para o modelo de PLI
        ils.addTerm(variavel, 1.0); // Adiciona o termo a expressão
        numérica CPLEX
    end
end
restricoes.add( $M.addEq(ils, 1)$ ) // Adiciona restrição ao modelo de PLI
return  $restricoes$ 

```

fim

Algoritmo 22: Algoritmo que adiciona as restrições à modelagem matemático do PCC dirigido

Entrada: Grafo G' , **IloCplex** M , **IloLinearNumExpr** $funcao_objetivo$, $List<IloRange>$ $restricoes$, **IloNumVar**[] $variaveis$

início

IloLinearNumExpr ils // Variavel CPLEX para manipulação da expressão numérica

IloNumVar $variavel$ // Variavel CPLEX para uso no modelo de PLI

while *existirem vértices n ainda não visitadas em G'* **do**

$ils = M.linearNumExpr()$

if $deg^+(n) > deg^-(n)$ **then**

while *existirem vértices $nInc$ incidentes em n* **do**

 // Transforma a aresta $(nInc, n)$ em uma variável para o modelo de PLI

$variavel = TRANSFORMA_ARESTA_VARIABEL(nInc, n)$

$ils.addTerm(variavel, 1.0);$ // Adiciona o termo a expressão numérica CPLEX

end

 // Adiciona restrição ao modelo de PLI

$restricoes.add(M.addEq(ils, (deg^+(n) - deg^-(n))))$

end

if $deg^+(n) < deg^-(n)$ **then**

while *existirem vértices $nSuc$ ligados a n* **do**

 // Transforma a aresta $(n, nSuc)$ em uma variável para o modelo de PLI

$variavel = TRANSFORMA_ARESTA_VARIABEL(n, nSuc)$

$ils.addTerm(variavel, 1.0);$ // Adiciona o termo a expressão numérica CPLEX

end

 // Adiciona restrição ao modelo de PLI

$restricoes.add(M.addEq(ils, (deg^-(n) - deg^+(n))))$

end

end

return $restricoes$

fim

Algoritmo 23: Algoritmo para encontrar um caminho aumentado em um grafo

Entrada: $G = (V, E)$

 Matching M em G
Saída: Caminho aumentado em G

 ENCONTRAR_CAMINHO_AUMENTADO(G, M)

início

 Floresta = \emptyset

 Desmarcar todos vértices e arestas em G

 Marcar todas as arestas em M
for *todos vértices expostos* v **do**

 | Floresta += árvore $\{v\}$
end
while *existir vértices v não marcadas em Floresta com $\text{dist}(v, \text{raiz}(v))$ par* **do**
while *existir arestas $e = (v, w)$ não marcadas* **do**
if w *não está em Floresta* **then**

 | x = vértice emparelhado com w em M adicionar arestas (v, w) e (w, x) à
 | árvore de v
end
if w *está em Floresta* **then**
if $\text{dist}(w, \text{raiz}(w))$ *é par* **then**
if $\text{raiz}(v) \neq \text{raiz}(w)$ **then**

 | $P = \text{caminho}(\text{raiz}(v), v) \cup (w, \text{raiz}(w))$

 | **return** P
end
if $\text{raiz}(v) = \text{raiz}(w)$ **then**

 | B = o blossom formado por e e as arestas no caminho (v, w) em T

 | $G' = \text{contrair } G \text{ em } B$

 | $M' = \text{contrair } M \text{ em } B$

 | $P' = \text{ENCONTRAR_CAMINHO_AUMENTADO}(G', M')$

 | $P = P'$ expandido em G

 | **return** P
end
end
end

 marcar aresta e
end

 marcar vértice v
end
return *caminho vazio*
fim
