UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CARACTERIZAÇÃO TEÓRICA E LIMITES ASSINTÓTICOS PARA O PROBLEMA DA GERAÇÃO DE REDES CANDIDATAS NA BUSCA POR PALAVRAS-CHAVE EM BANCOS DE DADOS RELACIONAIS

JOÃO GUILHERME ALVES MARTINEZ

Julho de 2019

Manaus - AM

JOÃO GUILHERME ALVES MARTINEZ

CARACTERIZAÇÃO TEÓRICA E LIMITES ASSINTÓTICOS PARA O PROBLEMA DA GERAÇÃO DE REDES CANDIDATAS NA BUSCA POR PALAVRAS-CHAVE EM BANCOS DE DADOS RELACIONAIS

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas (PPGI/IComp, UFAM) como parte dos requisitos necessários à obtenção do título de Mestre em Informática .

Orientadora: Rosiane de Freitas Rodrigues, PhD.

Co-orientador: Altigran Soares da Silva, PhD.

Julho de 2019

Manaus - AM

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Martinez, João Guilherme Alves
M385c Caracterização Teórica e Limites Assintóticos para o Problema da Geração de Redes Candidatas na Busca por Palavras-Chave em Bancos de Dados Relacionais / João Guilherme Alves Martinez. 2019
77 f.: il. color; 31 cm.

Orientadora: Rosiane de Freitas Rodrigues
Coorientador: Altigran Soares da Silva
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Redes Candidatas. 2. Busca por Palavras-Chave. 3. Banco de Dados. 4. Complexidade Computacional. 5. Teoria dos Grafos. I. Rodrigues, Rosiane de Freitas II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Caracterização teórica e limites assintóticos para o problema da geração de redes candidatas na busca por palavras-chave em bancos de dados relacionais"

JOÃO GUILHERME ALVES MARTINEZ

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Profa. Rosiane de Freitas Rodrigues - PRESIDENTE

Prof. Edleno Silva de Moura - MEMBRO INTERNO

Prof. Fábio Protti - MEMBRO EXTERNO

Manaus, 10 de Julho de 2019

À minha família.

Agradecimentos

Quero agradecer a Deus por toda a trajetória percorrida até aqui. À minha orientadora professora Rosiane e ao meu co-orientador professor Altigran por todo apoio, paciência e compreensão. Agradeço também o apoio do professor Fábio Protti da UFF. Agradeço também à minha esposa e à minha mãe que ajudaram muito nesse processo de conciliação do estudo acadêmico com o trabalho profissional. E por fim agradeço aos colegas e demais professores do PPGI e aos colegas de trabalho do CTIC da UFAM.

Resumo da Dissertação apresentada ao PPGI/IComp/UFAM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

CARACTERIZAÇÃO TEÓRICA E LIMITES ASSINTÓTICOS PARA O PROBLEMA DA GERAÇÃO DE REDES CANDIDATAS NA BUSCA POR PALAVRAS-CHAVE EM BANCOS DE DADOS RELACIONAIS

JOÃO GUILHERME ALVES MARTINEZ

Julho/2019

Orientadora: Rosiane de Freitas Rodrigues, PhD.

Co-orientador: Altigran Soares da Silva, PhD.

A busca em coleções de dados utilizando consultas por palavras-chave (KwS) é um importante campo de estudo dentro da Ciência da Computação, que facilita e acelera a obtenção de informações relevantes. Entretanto, se por um lado houve um grande avanço nas buscas por palavras-chave em páginas web, que são bases heterogêneas e em sua maioria semi-estruturadas, nos bancos de dados relacionais ainda predomina a consulta estruturada elaborada com linguagens de consultas como a SQL. Tais bancos de dados não permitem que sejam feitas consultas por palavras-chave de forma direta, pois é necessário conhecer o esquema que organiza as tabelas e ter conhecimento técnico em tais linguagens, fatores que dificultam o processo de busca. Entretanto, já existem sistemas que utilizam palavras-chave da consulta e, de modo transparente ao usuário, o esquema do banco de dados relacional, para então montar internamente consultas SQL adequadas que retornem as respostas relevantes para os usuários (R-KwS). Tais consultas em SQL geradas são chamadas de Redes Candidatas. O problema com tal abordagem é que o enfoque de tais sistemas tem sido empírico e experimental, onde ressaltam uma dificuldade e demora na resolução do problema de Geração das Redes Candidatas (CNGP). Dado esse contexto, propõe-se nesta pesquisa uma caracterização teórica de sistemas R-KwS, modelando-os como uma combinação de dois problemas clássicos em teoria dos grafos, que são o problema da Árvore de Steiner e o problema de Coloração de Arestas,

denominado de problema da Árvore de Steiner com Arestas Coloridas (STCEP). É apresentada uma prova de NP-completude para o caso geral onde se considera um número arbitrário de palavras-chave na busca, fornecendo-se um algoritmo NP e realizando-se uma redução polinomial do problema da Cobertura Exata por 3-Conjuntos (X3C) para o STCEP. Adicionalmente, também foram determinados limites polinomiais assintóticos para o caso prático quando se considera apenas um pequeno número fixo de palavras-chave. Diferentemente do que a literatura até então assumia como verdade do caso geral, é demonstrado que, na prática, a dificuldade em termos de complexidade computacional não ocorre. Como resultados complementares, foi realizada a primeira implementação do algoritmo de melhor razão de aproximação para o problema da geração de todas as árvores geradoras mínimas, o algoritmo de Eppstein [12], seguida da primeira implementação do algoritmo exato para o problema clássico da árvore de Steiner de atraso polinomial para um número fixo de terminais, proposto por Dourado, Oliveira e Protti [8] e que usa o algoritmo de Eppstein. Análises comparativas com os principais algoritmos implementados de ambos problemas, separadamente, foram também realizadas. Por fim, foi elaborada uma nova heurística para o problema clássico da árvore de Steiner, que apresenta desempenho competitivo na prática e que elimina o passo exponencial do algoritmo de Dourado, Oliveira e Protti [8]. Esses resultados respondem perguntas em aberto em sistemas R-KwS, bem como constituem contribuições para o problema da Árvore de Steiner em Teoria dos Grafos.

Palavras-chave: Redes Candidatas, Busca por Palavras-Chave, Banco de Dados, Complexidade Computacional, Teoria dos Grafos, Árvore de Steiner, Árvore Geradora Mínima.

Abstract of Dissertation presented to PPGI/IComp/UFAM as a partial fulfillment of the requirements for the degree of Master in Informatics.

THEORETICAL CHARACTERIZATION AND ASYMPTOTIC BOUNDS FOR THE CANDIDATE NETWORK GENERATION PROBLEM IN KEYWORD SEARCH OVER RELATIONAL DATABASES

JOÃO GUILHERME ALVES MARTINEZ

July/2019

Advisor: Rosiane de Freitas Rodrigues, PhD.

Co-advisor: Altigran Soares da Silva, PhD.

The search in collections of data using keyword queries (KwS) is an important field of study within Computer Science, which facilitates and accelerates the obtaining of relevant information. However, if on the one hand there was a great advance in the keywords search in web pages, which are heterogeneous bases mostly semi-structured, in relational databases still dominates the structured query elaborated with query languages such as SQL. Such databases do not allow queries for keywords directly, because it is necessary to know the schema that organizes the tables and to have technical knowledge in such languages, factors that make the search process difficult. However, there are already systems that use query keywords and, transparently to the user, the relational database schema, to then internally mount appropriate SQL queries that return relevant responses to users (R-KwS). Such generated SQL queries are called Candidate Networks. The problem with such an approach is that the focus of such systems has been empirical and experimental, highlighting a difficulty and high processing time in solving the Candidate Network Generation problem. Given this context, it is being proposed in this research a theoretical characterization of an R-KwS system, modeling it as a combination of two classic problems in graph theory, which are the Steiner Tree problem and the Edge Coloring problem, called the Steiner Tree with Colored Edges problem (STCEP). In this way, the

NP-completeness proof is presented for the general case where an arbitrary number of keywords is considered in the search, providing an NP algorithm and performing the polynomial reduction from the Exact Cover by 3-Sets problem (X3C) to the STCEP. Additionally, asymptotic polynomial boundaries were also determined for the practical case, when only a small fixed number of keywords were considered. Contrary to what the literature up till now assumed to be true of the general case, it is demonstrated that in practice, the difficulty in terms of computational complexity does not occur. As a complementary result, the first implementation of the best approximation ratio algorithm for the problem of generation of all minimum spanning trees, the Eppstein [12] algorithm, was carried out, followed by the first implementation of the exact algorithm for the classical problem of the Steiner tree in graphs, of polynomial delay for a fixed number of terminals, proposed by Dourado, Oliveira and Protti [8], and that uses the algorithm of Eppstein. Comparative analysis with the main algorithms of both problems, separately, were also performed. Finally, a new heuristic was developed for the classic problem of the Steiner tree in graphs, which presented a competitive performance in practice and that eliminates the exponential step of the algorithm of Dourado, Oliveira and Protti [8]. These results answer open questions in R-KwS systems, as well as contributions to the Steiner Tree problem in Graph Theory.

Keywords: Candidate Networks, Keyword Search, Databases, Computational Complexity, Steiner Trees, Minimum Spanning Trees.

Sumário

Lista de Figuras **xii**

Lista de Tabelas **xv**

1 Introdução **1**

1.1 Hipótese de Pesquisa 3

1.2 Objetivos 3

1.2.1 Objetivo Geral 4

1.2.2 Objetivos Específicos 4

1.3 Principais Contribuições 4

1.4 Organização desta Dissertação 5

2 Fundamentos Teóricos **7**

2.1 Tópicos em Algoritmos e Complexidade Computacional 7

2.1.1 Notação Assintótica 7

2.1.2 Classes de Problemas 8

2.1.3 Algoritmos Exatos e Aproximados 9

2.1.4 Complexidade Parametrizada 10

2.2 Teoria dos Grafos e dos Conjuntos 12

2.2.1 Classes de Grafos 13

2.2.2 Árvores de Steiner 15

2.2.3 Cobertura Minimal 16

2.3 Recuperação de Informação e Banco de Dados 16

2.3.1 Modelo Relacional 19

2.3.2 Consultas Estruturadas do Tipo SQL 20

3 Busca por Palavras-Chave em Bancos de Dados Relacionais 22

3.1 Arquitetura Geral 24

3.2 Conceitos Básicos e Terminologia 25

3.3 Grafo Conjunto-Tupla 28

4 O Problema da Geração de Redes Candidatas 32

4.1 Modelagem Teórica do Problema 32

4.2 Modelagem do Problema em Grafos 35

5 Aspectos de Complexidade Computacional 40

5.1 Prova de NP-Completude para o problema STCEP 40

5.1.1 STCEP pertence à Classe de Problemas NP 40

5.1.2 STCEP pertence à Classe de Problemas NP-difíceis 41

5.2 Limites Assintóticos para o Cenário de Aplicação 45

5.2.1 Análise do Cenário Teórico 45

5.2.2 Análise do Cenário Prático 47

6 Algoritmos, Experimentos Computacionais e Análise Empírica 49

6.1 Análise e Implementação de Algoritmos para o Problema da Árvore de Steiner 49

6.1.1 Algoritmo de Takahashi e Matsuyama 50

6.1.2 Algoritmo de Dourado, Oliveira e Protti 51

6.1.3 Heurística baseada no Algoritmo de Dourado, Oliveira e Protti . . 54

6.2 Análise e Implementação de Algoritmos para a Geração de Todas as Árvores Geradoras Mínimas 57

6.2.1 Algoritmo de Yamada, Kataoka e Watanabe 58

6.2.2 Algoritmo de Wright 59

6.2.3 Algoritmo de Eppstein 62

6.2.4 Experimentos Computacionais Comparativos 66

6.3 Análise empírica para o STCEP como um R-KwS 69

7 Considerações Finais 71

Referências 73

Lista de Figuras

1.1 (a) Exemplo de um sistema de RI para busca em documentos. (b) Exemplo de sistema de Banco de Dados que recebe consultas do tipo SQL. (c) Exemplo de um sistema de RI para busca por palavras-chave em Bancos de Dados relacionais. 2

2.1 Representação gráfica de um grafo simples não-direcionado com 4 vértices e 5 arestas e de um multigrafo não-direcionado com 4 vértices e 7 arestas. 12
2.2 Uma representação gráfica para um caminho, um ciclo, um grafo desconexo e uma árvore. 14
2.3 Um grafo simples e uma possível árvore geradora. 14
2.4 Um grafo simples com custo total igual a 10 e uma possível árvore geradora mínima com custo igual a 5. 14
2.5 Um grafo exemplo com os terminais marcados em preto e ao lado uma árvore de Steiner mínima com um vértice de Steiner marcado em cinza. . 16
2.6 Um exemplo de arquitetura de um sistema clássico de RI. 18
2.7 Esquema de banco de dados de uma locadora de carros. 21

3.1 As duas categorias de sistemas de busca por palavras-chave em bancos de dados relacionais. 24
3.2 Arquitetura de um sistema de busca por palavras-chave em bancos de dados relacionais baseado em *schema graph*. 25
3.3 Esquema do banco de dados do IMDb utilizado por Coffman. 25
3.4 Rede de junção de tuplas formada pela consulta de exemplo *denzel, washington, gangster*. 26

3.5 Rede de junção de tuplas minimal e total formada pela consulta de exemplo *denzel, washington, gangster*. 26
3.6 Rede de junção de conjuntos-tupla formada pela consulta de exemplo *denzel, washington, gangster*. 27
3.7 Rede Candidata formada pela consulta de exemplo *denzel, washington, gangster*. 28
3.8 Grafo G_{CT} para a consulta *denzel, washington, gangster* no banco de dados IMDb. 29
3.9 Duas árvores retiradas de G_{CT} que também são RCs. 30
3.10 Subgrafo de G_{CT} (a) e quatro árvores derivadas (b). 31

4.1 Grafo conjunto-tupla G_{CT} para a consulta *denzel, washington, gangster* no banco de dados IMDb. 34
4.2 Cinco possíveis RCs para o *match* $M = \{\text{CAST}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$. 35
4.3 Exemplo de grafo com coloração de arestas. 36
4.4 Exemplo de coloração de arestas utilizando o mínimo possível de cores. . 36
4.5 (a) Grafo com arestas coloridas e terminais. (b) Árvore de Steiner com arestas coloridas inválida. (c) Árvore de Steiner com arestas coloridas válida. 37
4.6 (a) Grafo G_{TS} de entrada do CNGP. (b) Grafo remodelado para STCEP. . 38

5.1 Grafo criado a partir de instância do X3C. 43
5.2 (a) Exemplo de grafo com terminais em vermelho e (b) Instância equivalente com função de coloração para as arestas. 44

6.1 Exemplo de camadas do algoritmo entre dois vértices separados por $d_G = 4$. 53
6.2 (a) o grafo original com os terminais marcados em vermelho. (b) o grafo G_{RW} de menor custo. (c) duas árvores de Steiner resultantes. 54
6.3 (a) Grafo original de exemplo. (b) Árvore geradora mínima arbitrária. (c) Arestas candidatas a serem substitutas de (C,D). (d) Nova árvore geradora mínima a partir da troca pela substituta (B,D). 60
6.4 Pequeno exemplo de aplicação de uma *sliding operation*. 62

6.5 (a) Árvore T e ao lado somente os vértices para início da construção de B. (b) Seleção das menores arestas incidentes e formação dos novos vértices em B. (c) Seleção da menor aresta entre as árvores remanescentes e finalização de B. 65

Lista de Tabelas

2.1 Número de coberturas minimais para $n \leq 7$. 17

2.2 Tabela com dados de alunos de uma instituição. 20

3.1 Comparativo entre os principais trabalhos. 24

4.1 Exemplo de todos os *matches* para duas possíveis coberturas totais e minimais. 35

6.1 Comparativo entre alguns algoritmos para resolução da árvore de Steiner. O parâmetro l representa o número de folhas na árvore ótima, α o número de árvores de Steiner enumeradas, n o número de vértices do grafo e k o número de terminais a serem interconectados. 50

6.2 Qualidade das soluções encontradas utilizando a heurística proposta. . . . 57

6.3 Algoritmos para gerar árvores geradoras e mínimas. 58

6.4 Resultados para grafos completos com mesmo peso nas arestas.* tempo muito alto de execução. 68

6.5 Resultados para grafos aleatórios com $m = 3n$ e peso aleatório pras arestas. 69

6.6 Resultados para grafos completos com peso das arestas aleatórios. * tempo alto de execução. 70

Lista de Algoritmos

1 Reconhecimento de solução para o STCEP 41

2 Takahashi e Matsuyama minimum Steiner-tree 50

3 Dourado, Oliveira e Protti enumerate Steiner min-trees 52

4 Heurística para árvore de Steiner 55

5 Yamada, Kataoka e Watanabe all MSTs 59

6 Wright all MSTs 61

7 Operação de sliding 63

8 King Branching Tree 64

9 Verifica Aresta 66

10 Todas as spanning trees 67

11 Eppstein all MSTs 68

Capítulo 1

Introdução

A busca por palavras-chave em documentos é um campo abrangente de estudo na área de Recuperação de Informação dentro da Ciência da Computação e desperta grande interesse de organizações e usuários, pois facilita e acelera a busca por documentos relevantes. Esse tipo de busca ficou popularizado atráves de sites da Internet e isso desencadeou vários estudos visando resultados cada vez melhores, para que assim as organizações pudessem atingir cada vez mais pessoas interessadas em seus negócios de uma forma rápida, fácil e acessível.

Ao longo dos anos os algoritmos de Recuperação de Informação foram otimizados para melhor responder a consultas por palavras-chave de forma a cada vez mais perceber e extrairem informações que vão além das palavras digitadas pelo usuário, e assim garantir resultados de maior relevância. Como exemplo, é possível citar os algoritmos que fazem o uso dos links entre as páginas da Internet para criar uma classificação (*ranking*) entre os sites e definir aqueles com maior influência sobre os demais para alcançar respostas com maiores chances de responder corretamente a consultas submetidas por seus usuários [2].

Entretanto, se por um lado houve um grande avanço na busca por palavras-chave em documentos e páginas web, os bancos de dados relacionais do modelo tradicional ainda dependem de consultas estruturadas elaboradas com linguagens como a SQL (*Structured Query Language*). Tais bancos de dados não permitem que sejam feitas consultas por palavras-chave de forma direta, pois é necessário conhecer o esquema que organiza as tabelas e o usuário precisa ter algum conhecimento técnico em uma linguagem como a SQL, fatores esses que dificultam o processo.

Devido a essa necessidade, começaram a surgir estudos acerca de sistemas de busca

por palavras-chave em bancos de dados relacionais com diferentes abordagens para a resolução do problema ([1],[22],[20],[29]). Uma dessas abordagens consiste em receber como entrada as palavras-chave da consulta do usuário e o esquema do banco, para então montar consultas SQL adequadas que retornem as respostas relevantes para o usuário. As consultas em SQL geradas por tais sistemas são chamadas de **Redes Candidatas** ou ***Candidate Networks*** [21]. A Figura 1.1 mostra um exemplo de um sistema de RI tradicional, de um SGBD e de um sistema de busca por palavras-chave em bancos de dados relacionais.

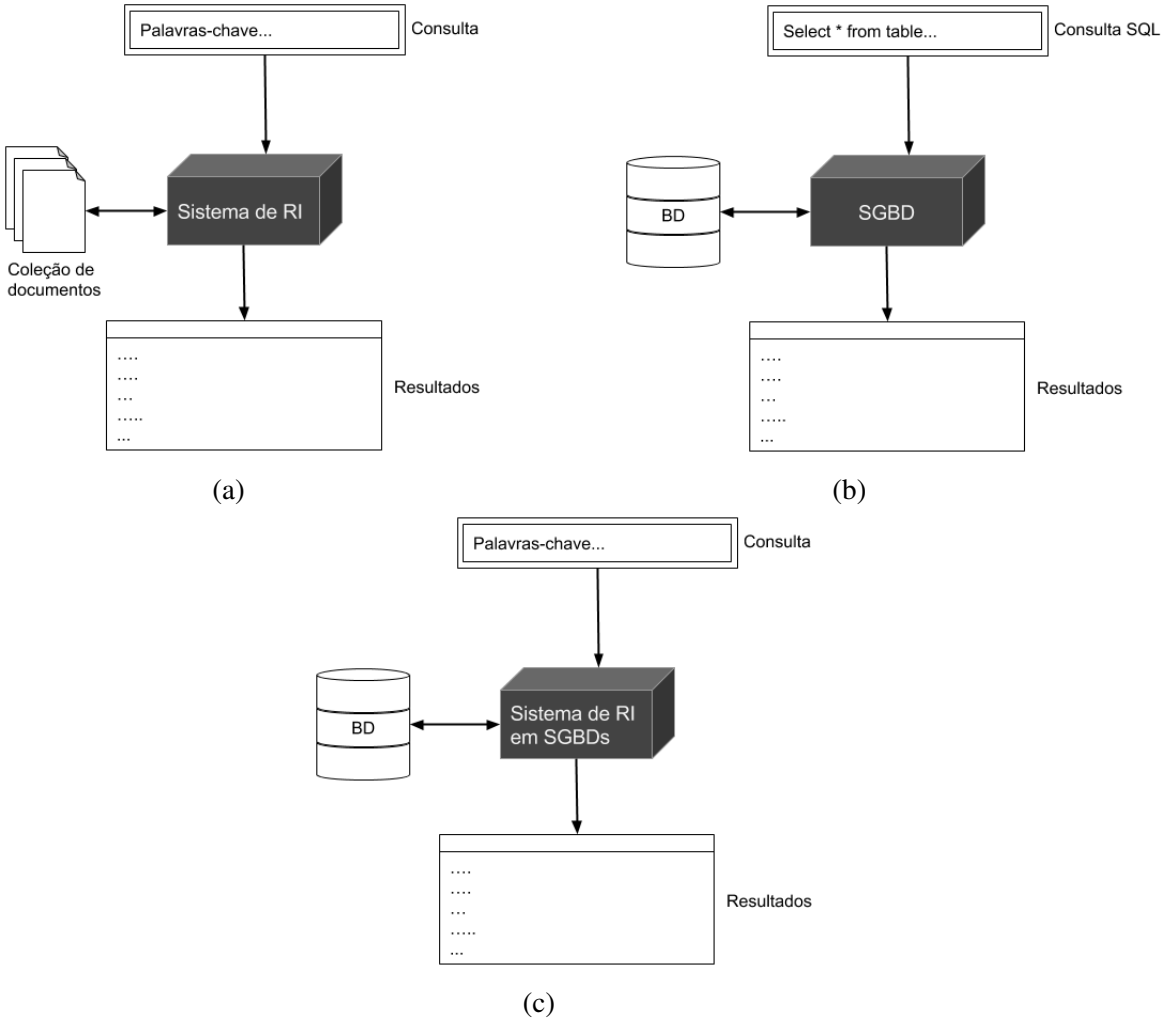


Figura 1.1: (a) Exemplo de um sistema de RI para busca em documentos. (b) Exemplo de sistema de Banco de Dados que recebe consultas do tipo SQL. (c) Exemplo de um sistema de RI para busca por palavras-chave em Bancos de Dados relacionais.

Atualmente existem diversos sistemas para o problema de busca por palavras-chave em bancos de dados relacionais que trabalham com o conceito de Redes Candidatas. Entretanto, o problema até então não possui uma modelagem teórica e todos os sistemas

propostos apresentam resultados empíricos e experimentais.

O problema possui importância prática e apresenta, portanto, desafios teóricos relevantes onde uma modelagem teórica do problema pode propiciar um melhor entendimento dos pontos de complexidade que possam permitir o desenvolvimento de melhores estratégias algorítmicas, além de fornecer indicadores teóricos matemático-computacionais e não somente experimentais e mostrar que, na prática, a dificuldade computacional esperada não ocorre.

1.1 Hipótese de Pesquisa

Dado o contexto apresentado anteriormente, duas importantes questões a se fazer neste cenário prático são:

1 É possível provar que o problema é NP-difícil?

2 Será que, considerando-se as restrições oferecidas no caso prático de aplicação, é possível que se tenha resultados com custo polinomial?

Com base nesta análise é levantada a seguinte hipótese de pesquisa: uma modelagem teórica do problema de busca por palavras-chave em bancos de dados relacionais pode definir os possíveis pontos de dificuldade computacional existentes e mostrar que mesmo que, no caso geral, o problema seja NP-díficil, na prática pode ser que as restrições da aplicação prática levem a situações de custo polinomial.

Desse modo, com base nesta hipótese levantada, foram definidos os objetivos a serem atingidos, conforme apresentados na próxima seção.

1.2 Objetivos

Com base nas questões de pesquisa levantadas e na hipótese que está se assumindo e se deseja comprovar, foram estabelecidos os objetivos geral e específico desta dissertação, que são descritos a seguir.

1.2.1 Objetivo Geral

O objetivo principal deste trabalho consiste em definir limites assintóticos computacionais para o problema prático da geração de Redes Candidatas na busca por palavras-chave em bancos de dados relacionais, através de uma caracterização teórica do problema e determinação de aspectos de complexidade computacional.

1.2.2 Objetivos Específicos

Para dar suporte ao objetivo geral descrito, seguem-se os objetivos específicos deste trabalho:

- investigar os principais sistemas de busca por palavras-chave em bancos de dados relacionais baseados em Redes Candidatas;
- modelar teoricamente o problema de geração de Redes Candidatas com base em problemas computacionais clássicos;
- analisar empricamente e provar teoricamente a complexidade computacional do problema da geração de Redes Candidatas;
- analisar a influência dos parâmetros na complexidade computacional geral do problema da geração de Redes Candidatas;
- investigar, implementar e analisar algoritmos para resolução do problema da árvore de Steiner em grafos.

1.3 Principais Contribuições

Como principais contribuições desta dissertação, tem-se:

- Modelagem teórica do problema de geração de redes candidatas (CNGP), na busca por palavras-chave em banco de dados relacionais (R-KwS);
- Proposição de um novo problema em grafos para modelar a aplicação citada no item anterior que consiste em uma variação do problema clássico de árvore de Steiner em grafos combinado com o problema clássico de coloração de arestas em grafos,

sendo denominado **problema de árvore de Steiner com arestas coloridas** (STCEP);

- Prova de NP-completude do STCEP, bem como análise de complexidade de pior caso e análise de complexidade de casos práticos relacionados à aplicação em banco de dados, com estabelecimento de limites assintóticos;
- Estudo, implementação e análise comparativa dos principais algoritmos para o problema clássico de árvores de Steiner;
- Estudo, implementação e análise comparativa dos principais algoritmos para o problema clássico de geração de todas as árvores geradoras míninas (usado nos principais algoritmos para o problema de árvore de Steiner em grafos);
- Implementação do algoritmo de melhor razão de aproximação para o problema da geração de todas as árvores geradoras mínimas, o algoritmo de Eppstein [12], seguida da implementação do algoritmo exato para o problema clássico da árvore de Steiner de atraso polinomial para um número fixo de terminais e que usa o algoritmo de Eppstein, proposto por Dourado, Oliveira e Protti [8];
- Proposição de uma heurística para o problema clássico da árvore de Steiner, que simplifica o passo exponencial do algoritmo exato robusto de Dourado, Oliveira e Protti [8], e que apresentou resultados competitivos com tempo polinomial de execução.

1.4 Organização desta Dissertação

O restante deste documento está organizado como segue. No Capítulo 2, sobre os fundamentos teóricos, são apresentadas as definições e conceitos de Teoria dos Grafos e dos Conjuntos, Complexidade Computacional, Recuperação de Informação e Banco de Dados. No Capítulo 3, é apresentado o problema da busca por palavras-chave em bancos de dados relacionais, os principais trabalhos relacionados e os conceitos e definições a partir de trabalhos relevantes da área. No Capítulo 4, é apresentada a modelagem teórica do problema da geração de Redes Candidatas e o mapeamento do mesmo para o problema da árvore de Steiner em grafos com arestas coloridas. No Capítulo 5 é demonstrada a

prova de NP-completude do problema da árvore de Steiner com arestas coloridas a partir do problema da cobertura exata de 3-conjuntos (X3C). Também é apresentada a análise do pior caso para o problema da geração de Redes Candidatas e uma análise para o caso prático. No Capítulo 6 são descritos os algoritmos desenvolvidos para resolução de árvores de Steiner, árvores geradoras mínimas e os respectivos experimentos. E por fim, no Capítulo 7, será dado sobre as considerações finais, são feitos os comentários gerais sobre a pesquisa, os resultados alcançados e trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

Neste capítulo são apresentados os conceitos que formam a base para o desenvolvimento deste trabalho, envolvendo os principais pontos teóricos em Algoritmos e Complexidade Computacional, Teoria dos Grafos e dos Conjuntos, Recuperação de Informação e Banco de Dados.

2.1 Tópicos em Algoritmos e Complexidade Computacional

De maneira informal, um algoritmo é definido como um procedimento computacional bem definido que toma um conjunto de valores como entrada e produz um conjunto de valores de saída [5].

2.1.1 Notação Assintótica

A análise de um algoritmo consiste numa avaliação dos passos a serem executados para a predição matemática acerca do desempenho esperado de processamento do algoritmo, independente dos dados a serem processados. O resultado dessa análise é uma função matemática que representa o limite assintótico do algoritmo, ou seja, uma função que representa como se comporta o tempo de execução do algoritmo a partir do crescimento dos seus dados de entrada.

A eficiência assintótica de um algoritmo é importante para determinar, dentre algoritmos que resolvem o mesmo problema, aquele que vai apresentar melhor desempenho para

todas as entradas (exceto as muito pequenas). A notação assintótica busca principalmente analisar uma função quando seus valores tendem ao infinito ou a valores muito altos.

Dadas as funções $f(n)$ e $g(n)$, a notação assintótica diz que $f(n)$ está na ordem de $g(n)$, ou que $f(n) = O(g(n))$, se $0 \leq f(n) \leq c.g(n)$, para alguma constante c positiva e para todo n suficientemente grande. Essa notação com a letra O grande, é chamada de **grande-O**, e representa o **limite superior** de uma função, ou seja, por maior que seja o tamanho de n para $f(n)$, nunca irá ultrapassar a função $g(n)$ para o mesmo valor de n. A notação do grande-O é muito utilizada para representar o desempenho de um algoritmo em seu pior caso de execução. Como exemplo, é possível dizer que a função $7n$ possui n^2 como limite superior, mas não necessariamente o contrário.

No entanto, a entrada para um algoritmo nem sempre coincide com o seu pior caso, sendo os dados de entrada muitas vezes similares para grande parte dos casos comuns. A esses casos deu-se o nome de caso médio, e a notação assintótica utilizada para representá-lo é o Θ (theta). Ou seja, dizemos que $f(n) = \Theta(g(n))$, se $c_1.g(n) \leq f(n) \leq c_2.g(n)$, para quaisquer constantes c_1 e c_2 positivas e para todo n suficientemente grande.

A outra notação é para o limite inferior, usada para representar o desempenho dos algoritmos no melhor caso, ou seja, o melhor desempenho possível do algoritmo. É dito que $f(n) = \Omega(g(n))$, se $0 \leq c.g(n) \leq f(n)$, para qualquer constante c e para todo n suficientemente grande.

Como exemplo de notação assintótica, o algoritmo *quicksort*, usado para ordenação de dados, possui limites assintóticos diferentes para cada caso. Em seu pior caso, apresenta um desempenho quadrático n^2, mas na maior parte dos casos, incluindo seu melhor caso, apresenta desempenho quase linear igual a $n \log n$, assim, nas três notações fica representado como $O(n^2)$, $\Theta(n \log n)$ e $\Omega(n \log n)$.

2.1.2 Classes de Problemas

Um problema é dito como **polinomial** quando o seu limite superior assintótico é representado por um polinômio. Isso garante que o algoritmo, por maior que seja o tamanho da entrada, possui um tempo de execução aceitável. No entanto, existem diversos problemas computacionais cujos algoritmos apresentam funções exponenciais e portanto são inviáveis para resolver instâncias de maiores tamanhos, pois mesmo com muitos recursos de hardware, precisariam de um tempo absurdamente grande para finalizar sua execução.

Com isso, os problemas foram classificados em classes de problemas. Os problemas da classe P são todos aqueles que podem ser resolvidos por pelo menos um algoritmo de tempo polinomial. Os problemas da classe NP são todos os problemas que são verificáveis em tempo polinomial, ou seja, dada uma possível solução, a verificação da corretude da solução é possível de ser feita em tempo polinomial. É possível perceber que todos os problemas em P também fazem parte de NP [14].

Uma outra classificação existente dos problemas são os **NP-difíceis**. Para ser classificado como NP-difícil, basta que o problema seja redutível a outro problema NP em tempo polinomial, independente de ser verificável em tempo polinomial ou não. Um problema é dito redutível a outro, caso exista um algoritmo que transforme a entrada do primeiro problema numa entrada equivalente para o segundo problema, de forma que a solução obtida no segundo problema possui equivalência com a solução buscada para o problema original.

Os problemas da classe **NP-Completos**, são todos os problemas que pertencem ao conjunto de problemas NP e ao conjunto de problemas NP-difíceis. O conceito de NP-Completo foi introduzido em 1971 por Stephen Cook, e a partir dele, vários outros problemas foram sendo provados também como NP-Completos.

Uma grande questão ainda em aberto é se $P = NP$ ou se $P \neq NP$. Caso seja provado que $P = NP$, então todos os problemas NP-Completos poderão ser resolvidos em tempo polinomial já que apresentam reduções polinomiais e são verificáveis em tempo polinomial. A hipótese contrária também não foi provada até o momento.

2.1.3 Algoritmos Exatos e Aproximados

Um algoritmo é dito como **exato**, se ao final de seu processamento oferece a solução ótima para o problema ao qual foi projetado para resolver. Uma solução é dita **ótima**, quando supera todas as outras soluções em termos de valor do resultado. Por exemplo, para o problema do caminho mínimo, uma solução é dita como ótima se apresenta uma rota com o menor caminho possível entre os dois pontos dados como entrada.

Muitos problemas importantes e aplicavéis no mundo hoje, são da classe NP-Completo e necessitam de solução para casos práticos, sem necessariamente obter a solução ótima, bastando-se de soluções que sejam o mais próximo possível da ótima. A partir dessa necessidade, surgiram os algoritmos aproximativos e as heurísticas.

Os algoritmos **aproximados** são chamados dessa forma pois retornam soluções aproximadas da ótima em tempo polinomial. Tais algoritmos fazem uso de heurísticas, que são regras e métodos aplicadas ao projeto de algoritmos para ajudar na resolução dos problemas. Um exemplo de heurística seria, durante o processamento de um algoritmo, o descarte de soluções improváveis baseado em algum padrão reconhecido nos dados de entrada.

Dependendo da heurística utilizada, a qualidade da solução apresentada pelo algoritmo pode ser empírica ou formal através de uma fórmula matemática. Quando um algoritmo aproximado apresenta essa prova formal matemática acerca da qualidade da solução que ele fornece, ele é chamado de algoritmo **aproximativo**. A qualidade de tais algoritmos é atestada por meio de uma razão de aproximação, que é calculada matematicamente sempre em referência à solução ótima esperada, seja ela de maximização ou de minimização. Por exemplo, um algoritmo aproximativo de razão 2 é dito 2-aproximativo pois sempre apresentará soluções que sejam no máximo duas vezes o valor da solução ótima.

2.1.4 Complexidade Parametrizada

Os problemas ditos como NP-difíceis são chamados assim porque se tornam exponencialmente mais difíceis de serem resolvidos conforme os parâmetros de entrada crescem. No entanto, se for possível verificar que o problema cresce em função de apenas um parâmetro k específico que pode ser controlado em vários casos práticos, portanto o problema pode ser resolvido em tempo polinomial para obter a solução ótima, desde que o parâmetro k esteja sob controle [37].

A teoria de complexidade parametrizada já havia sido explorada na literatura mas foi inicialmente proposta nesses termos por Downey e Fellows [9]. Um problema faz parte da classe FPT (*Fixed Parameterized Tractable*) se admite um algoritmo cuja complexidade não-polinomial é somente em função de um parâmetro k, ou seja, é executável em tempo $f(k).n^{O(1)}$. Sendo assim, o mais importante nessa teoria é descobrir qual o parâmetro que torna o problema computacionalmente difícil [6].

Existem algumas técnicas que são utilizadas para demonstrar que um problema é tratável por parâmetro fixo como por exemplo árvores de busca limitada, decomposição em coroa, aplicação de técnicas de otimização combinatória ou análise probabilística, dentre

outras.

O problema da cobertura de vértices é um dos 21 problemas NP-Completos estudados por Karp [25]. Em sua descrição, o problema questiona se, para um dado grafo G, existe um subconjunto S de vértices com k elementos de forma que todas as arestas de G possuam ao menos uma de suas extremidades em S. Existe um algoritmo FPT clássico para este problema que funciona a partir da simples observação a seguir. Considere que se escolha aleatoriamente um vértice v de G e coloque em S. É possível afirmar que todas as arestas incidentes a v já estão cobertas e que portanto podem ser removidas de G. A partir disso, o algoritmo funciona da seguinte maneira, escolhe uma aresta aleatória uv de G e faz chamadas recursivas com duas escolhas possíveis, inserindo u ou v em S. Ao escolher um vértice só se pode selecionar agora $k-1$ elementos para cobrir o restante do grafo, portanto, dada a instância inicial (G,k) do problema, as chamadas recursivas são $(G-u,k-1)$ e $(G-v,k-1)$. Note que se o parâmetro k chegar a zero e ainda existir arestas em G é porque não há solução. Do contrário, se o grafo não mais possuir arestas com $k \geq 0$ é porque existe solução. Este algoritmo trabalha explorando as possíveis soluções em uma árvore recursiva com desempenho $O(2^k)$ e demonstra a técnica de árvore de busca limitada [37].

Também um método importante para a construção de algoritmos FPT é a redução do algoritmo para um núcleo, técnica chamada de *kernelization*, que basicamente consiste em reduzir em tempo polinomial a instância inicial I do problema para uma instância I', tal que I' seja limitado por alguma função de k independente do tamanho da instância inicial I. Tal técnica pode ser entendida como um pré-processamento para eliminar partes da instância que são fáceis de serem resolvidas ou até mesmo descartar partes que não façam parte de nenhuma solução [37].

Um exemplo da redução de núcleo pode ser visualizada de forma simples no problema de cobertura de vértices através de observações simples que criam regras para a redução. Na regra 1, percebe-se que é possível retirar todos os vértices isolados do grafo G de entrada pois como não possuem arestas, estes nunca farão parte de qualquer solução possível. Na regra 2, percebe-se que um vértice com grau maior ou igual a $k+1$ deve ser colocado em qualquer solução do problema, caso contrário não será possível cobrir todas as arestas do problema com somente k vértices, sendo assim é possível retirar esse vértice e suas arestas adjacentes da instância e reduzir k em uma unidade e obter assim uma ins-

tância menor do problema $(G-v,k-1)$. Na regra 3, caso a instância após a aplicação das regras 1 e 2, satisfazer alguma das condições: $k<0$ ou G possuir mais de k^2+k vértices ou G possuir mais de k^2 arestas, é porque não possui solução [6].

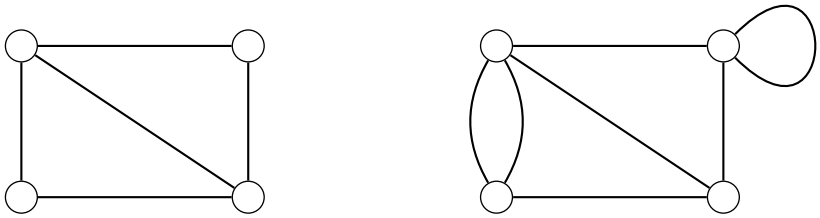
2.2 Teoria dos Grafos e dos Conjuntos

Esta seção formaliza a noção de grafos e conjuntos e seus conceitos básicos, bem como algumas de suas classes importantes, baseado no livro de Diestel [7]. As terminologias e notação utilizadas no restante deste documento são introduzidas nesta seção.

Os grafos constituem uma poderosa ferramenta capaz de modelar diversos tipos de relações e processos do mundo real, tendo aplicação nas áreas da biologia, química, computação, logística, dentre outras.

Um grafo $G=(V,E)$ com n vértices e m arestas consiste em um conjunto de vértices $V(G)=\{v_1,...,v_n\}$ e em um conjunto de arestas $E(G)=\{e_1,...,e_m\}$ no qual cada aresta é um par (u,v), indicando que o vértice u é adjacente ao vértice v. Um *loop* é uma aresta que possui os mesmos dois vértices como extremos. Arestas paralelas (ou múltiplas), são arestas que possuem o mesmo par de vértices. Um grafo é dito como simples quando não possui arestas paralelas, nem *loops*, caso contrário é chamado de **multigrafo**. Um grafo é classificado como não-direcionado, quando suas arestas não fazem distinção de vértice origem e destino, sendo u adjacente a v e vice-versa.

Na representação gráfica de um grafo, os vértices são colocados como pontos e as arestas são os arcos que conectam dois vértices do par. A disposição dos vértices e das arestas não altera a estrutura do grafo, pois trata-se somente de uma representação visual. Na Figura 2.1 é possível ver dois exemplos de grafos representados graficamente.



(a) (b)

Figura 2.1: Representação gráfica de um grafo simples não-direcionado com 4 vértices e 5 arestas e de um multigrafo não-direcionado com 4 vértices e 7 arestas.

A **ordem** de um grafo corresponde ao número de vértices que possui, enquanto que o **tamanho** de um grafo corresponde ao número de arestas que possui.

O grau $d(v)$ de um vértice v do grafo equivale ao número de arestas que incidem em v. O grau máximo de um grafo, detonado por $\Delta(G)$, é o valor do maior grau dentre todos os vértices de G. O grau mínimo de um grafo é denotado por $\delta(G)$, que também corresponde ao menor grau dentre todos os vértices de G.

2.2.1 Classes de Grafos

Nesta seção estão destacadas algumas importantes classes de grafos.

Definição 2.1. *Um **caminho** é uma sequencia de vértices $P = \{v_1, ..., v_n\}$ de forma que dois vértices só são adjacentes caso sejam consecutivos na sequência.*

O comprimento de um caminho P é $|P|$, e o número de arestas é sempre $|P| = n - 1$ [7]. Um exemplo de um caminho é mostrado na Figura 2.2a.

Definição 2.2. *Um **ciclo** C_n (para $n \geq 3$) é um grafo cíclico onde os vértices somente são adjacentes caso sejam consecutivos na sequência, e também o primeiro e último vértice coincidem.*

Um exemplo de um ciclo está representado na Figura 2.2b. Um grafo G é dito como **acíclico** quando não possui ciclos.

Definição 2.3. *Um grafo é dito como **conexo**, se dados dois vértices distintos u e v de G, existe pelo menos um caminho que os une. Um grafo que não é conexo é chamado de **desconexo**.*

A Figura 2.2c representa um grafo desconexo.

Definição 2.4. *Uma **árvore** é um grafo simples, conexo e acíclico.*

A Figura 2.2d representa uma árvore pois não possui ciclos e todos os vértices possuem ao menos um caminho para todos os outros.

Definição 2.5. *Uma **folha** é um vértice de grau igual a 1.*

Definição 2.6. *Uma **árvore geradora** é um subgrafo de G que é uma árvore e possui todos os vértices de G.*

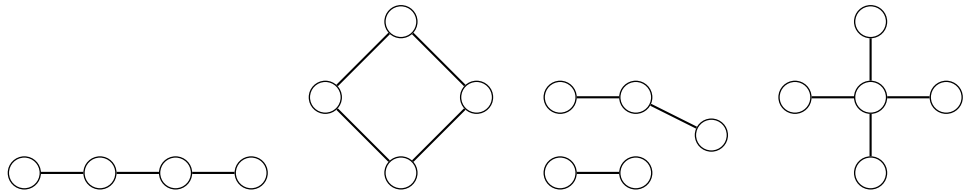


Figura 2.2: Uma representação gráfica para um caminho, um ciclo, um grafo desconexo e uma árvore.

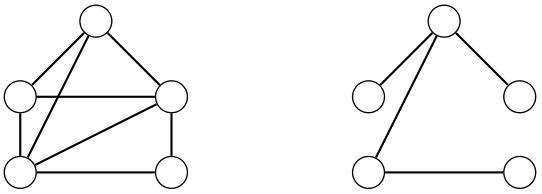


Figura 2.3: Um grafo simples e uma possível árvore geradora.

A Figura 2.3 representa um grafo e ao seu lado uma possível árvore geradora desse grafo.

Um grafo **ponderado** consiste em um grafo com valores numéricos associados a cada aresta. Isso permite a modelagem de vários problemas do mundo real, especialmente de otimização.

Definição 2.7. *Uma **árvore geradora mínima** consiste em uma árvore geradora de um grafo G, cuja soma dos pesos das arestas escolhidas é a menor possível.*

Encontrar uma árvore geradora mínima constitui assim um problema de otimização, pois busca-se um subgrafo do tipo árvore que minimize o peso total. A Figura 2.4 apresenta um exemplo de grafo ponderado e ao lado uma possível árvore geradora mínima.

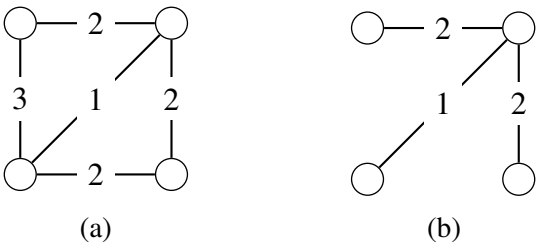


Figura 2.4: Um grafo simples com custo total igual a 10 e uma possível árvore geradora mínima com custo igual a 5.

O problema de encontrar uma árvore geradora mínima encontra-se na classe de problemas P, portanto pode ser resolvido em tempo polinomial. Os mais famosos algoritmos

desenvolvidos são os de Prim e Kruskal, que podem, dependendo da estrutura de dados que fazem uso, apresentar desempenho computacional da ordem de $O(m + n \log n)$, como na implementação de Prim com uso de heaps de fibonacci [12].

2.2.2 Árvores de Steiner

A primeira descrição do problema das árvores de Steiner, diz que: dado um plano com n pontos, uma árvore mínima de Steiner é uma árvore que interconecta tais pontos utilizando linhas de menor tamanho possível. No entanto, para se alcançar árvores de tamanho mínimo, a árvore de Steiner pode conter vértices adicionais aos originais. Vértices extras adicionados a árvore para reduzir seu custo são chamados de vértices de Steiner, e recebem esse nome graças a J. Steiner que foi o primeiro a trabalhar em cima do caso onde $n = 3$ [15]. Um dos primeiros algoritmos para encontrar árvores mínimas de Steiner foi proposto por Melzak [35], mas explora um grande número de possibilidades o que o torna viável computacionalmente somente para valores pequenos de n.

A segunda variação do problema ficou conhecida como árvore de Steiner *rectilinear*, semelhante ao primeiro problema, mas neste as distâncias entre os pontos deveriam ser calculadas a partir da diferença absoluta entre os pontos cartesianos. Isso implica que somente podem ser utilizadas linhas horizontais ou verticais para interligar os pontos. Tal problema possui forte aplicação prática no desenho de circuitos eletrônicos [16].

A terceira variação do problema ficou muito conhecida por ser aplicada diretamente em grafos: dado um grafo G e um subconjunto dos vértices de G chamados de **terminais**, encontrar uma árvore que conecte todos os vértices do conjunto de terminais, podendo ou não utilizar outros vértices de G [10]. Os demais vértices não-terminais que forem utilizados para criar a árvore são os vértices de Steiner conforme dito anteriormente. Um exemplo de árvore de Steiner em grafos é apresentado na Figura 2.5.

O problema da árvore de Steiner em grafos pode ser visto como uma generalização de outros dois problemas em teoria dos grafos muito conhecidos: caminho mínimo e árvore geradora mínima. Caso uma instância do problema possua somente dois terminais, a mesma se enquadra no problema do caminho mínimo. Caso em uma outra instância do problema, todos os vértices do grafo sejam terminais, o mesmo se enquadra no problema da árvore geradora mínima.

O problema de decisão foi classificado pela primeira vez como NP-Completo por

Karp [25]. Seu problema de otimização consiste em obter uma árvore que conecte todos os terminais que minimize o custo total das arestas escolhidas. Esse problema por sua vez, é NP-difícil e portanto não existe um algoritmo exato que o resolva de forma ótima em tempo polinomial, no entanto, já existem diversos algoritmos que fazem uso de heurísticas além de algoritmos aproximativos que obtêm soluções com boas taxas de aproximação ([28],[41],[44],[46]).

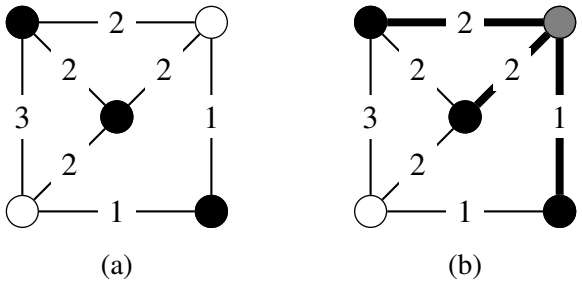


Figura 2.5: Um grafo exemplo com os terminais marcados em preto e ao lado uma árvore de Steiner mínima com um vértice de Steiner marcado em cinza.

2.2.3 Cobertura Minimal

Considere um conjunto *S* com *n* elementos, uma **cobertura** de *S* consiste em uma coleção de subconjuntos não-vazios de *S*, cuja união destes resulta em *S*. Uma cobertura **minimal**, no entanto, consiste em uma cobertura na qual a remoção de qualquer um de seus subconjuntos destrói a propriedade de cobertura [18].

Por exemplo, considere o conjunto{1,2} que possui cinco coberturas: {{1},{2}}, {{1,2}}, {{1},{1,2}}, {{2},{1,2}} e {{1},{2},{1,2}}. No entanto, de todos esses conjuntos, somente as coberturas {{1},{2}} e {{1,2}} são minimais, porque se qualquer um de seus subcojuntos for removido, a propriedade de cobertura não irá ser mantida dado que algum(s) membro(s) do conjunto original {1,2} irá faltar. A Tabela 2.1 apresenta o número de coberturas minimais contendo *k* subconjuntos, para conjuntos com *n* elementos.

2.3 Recuperação de Informação e Banco de Dados

Dentro da Ciência da Computação, a área de Recuperação de Informação (RI) tem como objetivo prover acesso à documentos de uma coleção que contenham as informações de

n \ k	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1						
2	1	1					
3	1	6	1				
4	1	25	22	1			
5	1	90	305	65	1		
6	1	301	3410	2540	171	1	
7	1	966	33621	77350	17066	420	1

Tabela 2.1: Número de coberturas minimais para $n \leq 7$.

interesse dos usuários de uma forma fácil e clara. Um dos focos de estudo da área, que é o abordado neste trabalho, é centrado no computador, na construção e desenvolvimento de estruturas e algoritmos para melhorar a qualidade e a velocidade dos resultados.

Desde os tempos antigos, a humanidade armazena informação que depois se torna objeto de busca. Isso era feito por meio de indexação, categorização, armazenamento em livros, papiros, dentre outros. Nesse processo surgiram as bibliotecas, que armazenam e provêm acesso a um enorme volume de informação. Para acelerar a busca por informações específicas, surgiram os índices, que são a base dos sistemas de RI. No princípio, os índices eram criados de forma manual para categorizar os documentos por tópicos e com o surgimento dos computadores, começaram a surgir maiores estudos para que sistemas pudessem ser criados para contribuir nessa área.

Os primeiros sistemas de RI eram somente para bibliotecas e não atraiam a atenção de pesquisadores que não fossem bibliotecários ou especialistas em informação. No entanto, o surgimento e popularização da Internet no início dos anos 90, possibilitaram que qualquer usuário criasse e publicasse seus documentos de forma fácil, rápida e barata, o que fez com que a Internet se tornasse o maior repositório público de informação. Consequentemente, a busca por informações em uma base de documentos desse tamanho não é fácil, e a partir dai aumentaram os esforços de pesquisadores para desenvolver sistemas de RI para a Internet que foram batizados como máquinas de busca.

Um usuário de um sistema de RI usualmente está à procura de alguma informação específica que é necessária para que ele resolva algum problema ou tarefa ou simplesmente por que deseja ter acesso aquela informação. No entanto, uma descrição textual e exata

da necessidade não é necessariamente a melhor entrada para um sistema de RI. A tradução dessa descrição da necessidade é chamada de **consulta** e usualmente é constituida somente pelas palavras-chave que melhor descrevem a informação desejada pelo usuário, e assim o objetivo do sistema é sempre trazer as informações que sejam úteis e relevantes para aquela necessidade específica do usuário. Tal busca consiste numa interpretação das informações sintáticas e semânticas do texto dos documentos da coleção que então classifica e ordena os documentos pela sua relevância. A noção de relevância é muito importante para um sistema de RI porque depende de um julgamento pessoal subjetivo que varia muito a depender das circustâncias como tempo, local e contexto [2].

Não se pode confundir Recuperação de Informação com recuperação de dados. De forma simples, a recuperação de dados somente visa trazer os documentos que possuem as palavras-chave solicitadas, sem uma análise da relevância das informações que possam satisfazer à consulta do usuário.

A Figura 2.6 ilustra a arquitetura básica de qualquer sistema de RI. Dois processos ocorrem em um sistema de RI mas que não necessariamente são simultâneos. No processo de indexação, os documentos da coleção são escaneados para que seja construída uma indexação que acelere o processamento das consultas posteriormente. A estrutura mais utilizada é o índice invertido que relaciona para cada palavra encontrada, uma lista de documentos nos quais ela aparece. Esse processo é executado de tempos em tempos caso a coleção de documentos seja dinâmica.

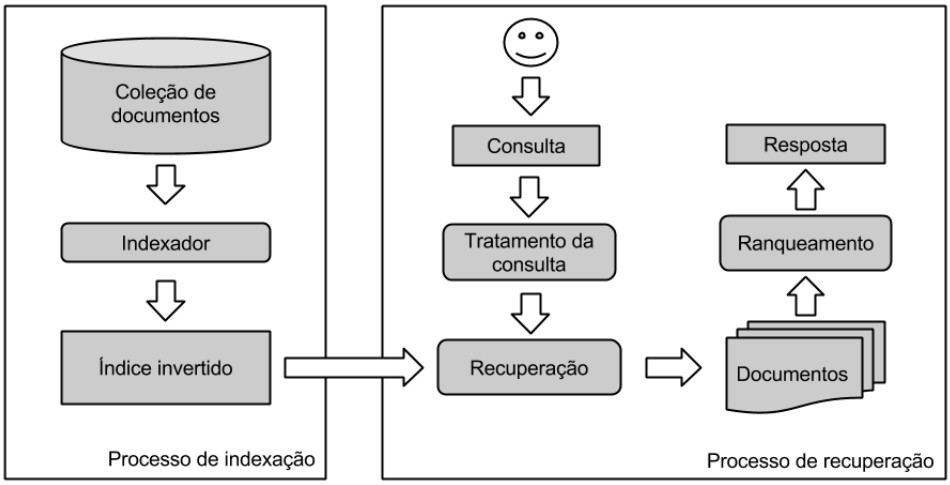


Figura 2.6: Um exemplo de arquitetura de um sistema clássico de RI.

O outro processo ocupa a parte central do sistema de RI, que consiste na busca e classificação da informação requisitada pelo usuário. Inicialmente o usuário fornece uma en-

trada textual, a consulta, que possui as palavras-chave. O sistema então faz um tratamento na consulta, que comumente engloba ações como correção ortográfica das palavras, adição de outras informações coletadas pelo sistema como por exemplo cliques, e remoção de *stopwords* (palavras irrelevantes como "de"e "para"). Então entra a parte central do sistema que consiste num método ou algoritmo bem definido para recuperar os documentos relevantes por meio do índice invertido e da consulta tratada. Com a lista de documentos considerados relevantes, entra a fase de ranqueamento onde os documentos são colocados em ordem decrescente de probabilidade de atenderam à consulta. Esses dois passos concentram grande parte do trabalho de inteligência de qualquer sistema de RI, e são alvo de estudos de otimização e melhoria. Por fim o sistema apresenta os documentos dados como resposta para o usuário.

2.3.1 Modelo Relacional

Os bancos de dados se tornaram um componente essencial em praticamente todos os sistemas utilizados diariamente, como por exemplo, transações monetárias, buscas por informação pela Internet, compra de produtos, dentre outros. Sendo assim, um banco de dados é definido como uma coleção de dados. Os dados são fatos ou informações que podem ser armazenadas e que possuem algum significado, como por exemplo nomes, números de telefone e endereços [11].

Um banco de dados pode ser feito de forma manual ou automatizada por meio de computador. Um catálogo físico de livros de uma biblioteca é um exemplo de um banco de dados manual. Um **sistema gerenciador de banco de dados (SGBD)** consiste de softwares que facilitam a criação e manipulação de bancos de dados computacionais. Tais sistemas permitem ao usuário definir os tipos de dados, sua estrutura, as restrições de integridade, além de controlar o armazenamentos desses dados na mídia física, manipular funções como busca, otimização e atualizações, gerenciar acessos múltiplos, concorrência e também possuem mecanismos de segurança contra acessos indevidos e perda de dados em casos de falhas físicas ou lógicas.

Os sistemas de banco de dados podem ser representados a partir de vários modelos diferentes, mas o mais utilizado até hoje pela maior parte dos sistemas é o modelo relacional, que representa o banco de dados como uma coleção de relações. Os dados são armazenados em tabelas definidas pelo projetista do banco de dados e cada registro de

uma tabela é chamado como **tupla**. Uma tupla de uma tabela possui vários dados que são separados por colunas na tabela. A Tabela 2.2 mostra um exemplo de uma tabela com dados de alunos de uma instituição.

ID	Nome	Data_Nascimento	Endereço
5	Francisco Moura	04/12/1981	Rua dez, 48, Copacabana
6	Maria das Graças	03/09/1973	Av. França, 31, Botafogo
..	..	..	..

Tabela 2.2: Tabela com dados de alunos de uma instituição.

Cada tabela possui ao menos uma coluna cujos valores devem ser únicos, ou a junção dos valores de duas ou mais colunas deve ser único, a fim de que cada tupla de qualquer tabela possa ser identificada de maneira única. Esse valor único de cada tupla é chamado de **chave primária** da tabela. No exemplo da Tabela 2.2, a chave primária é a coluna ID.

Com as chaves primárias, é possível criar restrições de integridade entre as tuplas ao criar colunas nas tabelas para abrigar o valor da chave primária de outras tabelas. Ainda de acordo com o exemplo da Tabela 2.2, imagine uma outra tabela chamada CURSO, com as colunas ID_Curso e Nome_Curso, onde a coluna ID_Curso é a chave primária. É possível adicionar uma nova coluna na tabela ALUNO com o nome ID_Curso e nela inserir, para cada aluno, a chave primária do seu curso na instituição, assim pode-se obter tanto os dados do aluno, como os dados do curso ao qual está relacionado. Esse valor de chave primária que é usado como referência em outra tabela, é chamado de **chave estrangeira**. Uma chave estrangeira sempre é uma chave primária de outra tabela.

O conjunto de tabelas, colunas e restrições de integridade por meio de chaves é chamado de **esquema do banco de dados**. Esse esquema é sempre único, devendo ser projetado por um especialista que modela os dados do mundo real que serão armazenados computacionalmente. A Figura 2.7 mostra um exemplo de um esquema de banco de dados de uma locadora de carros.

2.3.2 Consultas Estruturadas do Tipo SQL

Para manipular os dados no modelo relacional, foi criada uma linguagem específica chamada *Structured Query Language* (SQL). Com ela, é possível criar e definir tabelas, colunas, restrições, regras, inserir dados e também alterar e excluir. Possui uma sintaxe

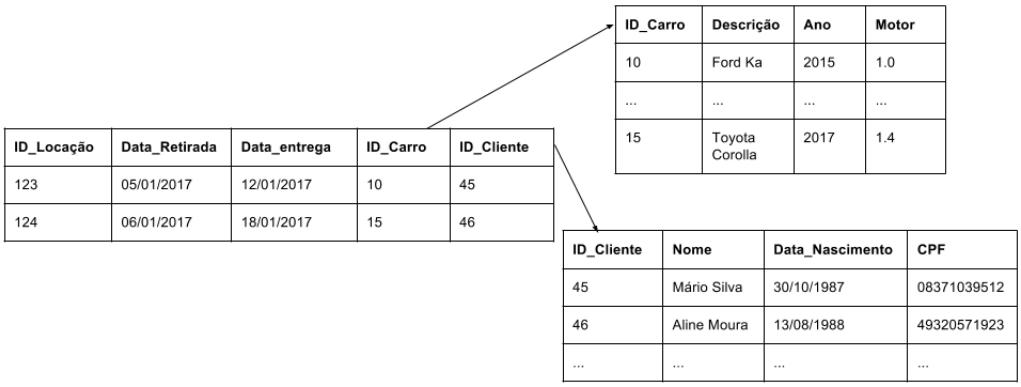


Figura 2.7: Esquema de banco de dados de uma locadora de carros.

genérica que é suportada em sua maior parte por quase todos os SGBD's comerciais existentes.

A SQL pode ser dividida em duas: a DDL, *Data Definition Language*, que são os comandos usados para definir os dados, ou seja, criar tabelas, colunas, restringir o tipo dos dados, especificar regras para cada coluna, dentre outras, e também na DML, *Data Manipulation Language*, que possui os comandos para manipular os dados no banco de dados, ou seja, inserir, alterar, buscar e excluir [11].

Neste trabalho é feito uso somente da DML da SQL e mais especificamente do comando SELECT, utilizado para buscar tuplas nas tabelas do banco de dados. Um exemplo de uma consulta SQL baseado no esquema da Figura 2.7 que obtem o cliente Mário Silva que comprou um Ford Ka: *SELECT * FROM Locacao INNER JOIN Clientes ON Clientes.ID_Cliente = Locacao.ID_Cliente INNER JOIN Carros ON Locacao.ID_Carro = Carros.ID_Carro WHERE Clientes.Nome Like "%Mario Silva%"AND Carros.Descricao Like "%Ford Ka%"*.

Capítulo 3

Busca por Palavras-Chave em Bancos de Dados Relacionais

Os atuais sistemas de busca por palavras-chave em bancos de dados relacionais são divididos em duas categorias: sistemas baseados em *schema graphs* (grafos do esquema) e sistemas baseados em *data graphs* (grafos de dados). Os sistemas da primeira categoria são baseados no conceito de **Redes Candidatas** (*Candidate Networks*), que são redes de tuplas interconectadas usadas para gerar consultas SQL cujos resultados irão responder à consulta do usuário. As vantagens dessa abordagem consiste no pouco uso de memória devido ao grafo usualmente ser pequeno e também porque tais sistemas se apropriam das funcionalidades básicas do sistema gerenciador de banco de dados relacional (SGBD) para produzir as consultas em SQL. A desvantagem é que tais sistemas por vezes fazem buscas genéricas que podem trazer muitos resultados irrelevantes. A seguir será comentado sobre alguns sistemas que se enquadram nessa categoria.

O sistema DISCOVER [21] cria um índice *master* que mapeia as ocorrências das palavras-chave nas tabelas do banco de dados sem preocupar-se com as colunas onde as palavras aparecem. Em seguida utiliza um algoritmo guloso de busca em largura para explorar o *schema graph* e criar as Redes Candidatas (RCs), que são então convertidas em consultas do tipo SQL. Os mesmos autores posteriormente trabalharam em melhorias com a aplicação de técnicas de RI tradicionais para melhorar os resultados [20].

O trabalho de Oliveira [36] também baseia-se nos conceitos e algoritmo colocados em DISCOVER [21], mas particiona o algoritmo em dois passos menores, onde no primeiro faz todas as possíveis combinações de subconjuntos de palavras para realizar, por subcon-

junto, a busca em largura no grafo do esquema. Também fez melhorias no algoritmo de ranqueamento das RCs.

Em DBXplorer [1] é criada uma tabela de símbolos que pode ter dois níveis de granularidade: de célula ou de coluna. A nível de célula, a tabela armazena para cada palavra, a exata linha onde ocorreu, já a nível de coluna a tabela somente armazena em qual coluna da tabela apareceu a palavra. O sistema também enumera árvores a partir do grafo do esquema fazendo a junção das tabelas onde ocorreram as palavras-chave considerando as possibilidades de árvores que contenham todas as palavras da consulta.

Outro sistema que também faz uso de *schema graph* é o SPARK [29], mas que tem como foco o ranqueamento das RCs, que são extraídas pelo mesmo método e algoritmo de DISCOVER [21].

Os sistemas da segunda categoria são baseados em estruturas chamadas de *data graphs*, que são grafos cujos vértices representam as tuplas que contêm as palavras-chave da consulta e as arestas que conectam essas tuplas são baseadas nas chaves de integridade referencial. Nessa abordagem, o resultado de uma consulta é dado por subárvores do grafo, que minimizam a distância entre os vértices que contêm as palavras-chave. A vantagem de tais sistemas é a maior precisão para buscar as tuplas mais relevantes. As desvantagens são o grande espaço ocupado pelo grafo em memória e a necessidade deste ser atualizado sempre que alguma alteração for feita no banco de dados.

Um dos primeiros sistemas criados com essa estrutura foi BANKS [22]. Como já dito, cada tupla de cada tabela torna-se um vértice de um grande grafo que é mantido em memória. Nele é aplicado um algoritmo de busca expandida reversa com sucessivas aplicações do algoritmo de Dijkstra para o menor caminho.

O sistema BLINKS [17] também é baseado em *data graphs* e utiliza um algoritmo baseado no desenvolvido em BANKS [22], mas com algumas modificações, principalmente no quesito de avaliação para poder recuperar as melhores respostas primeiro.

Por fim, no trabalho de *Bidirectional search* [23], foi desenvolvido um algoritmo de busca bidirecional que segundo os autores pode ser aplicado em ambas as categorias, mas foi aplicado com uso de *data graphs* nesse trabalho em específico.

A Tabela 3.1 mostra um comparativo entre os principais trabalhos.

A Figura 3.1 demonstra essa divisão dos sistemas nas duas categorias. Neste trabalho serão explorados e investigados os conceitos para os sistemas da primeira categoria,

Sistema/Artigo	Ano	Categoria	Autores	Bases de testes
DISCOVER	2002	*Schema graph*	Hristidis e Papakonstantinou	TPC-H
DBXplorer	2002	*Schema graph*	Agrawal, Chaudhuri e Das	TPC-H, ML, USR e KB
BANKS	2002	*Data graph*	Hulgeri e Nakhe	DBLP
Efficient IR-Style K-Sw over Relational DBs	2003	*Schema graph*	Hristidis, Gravano e Papakonstantinou	DBLP
Bidirectional search	2005	*Data graph*	Kacholia, Pandit, Chakrabarti, Sudarshan, Desai e Karambelkar	DBLP, IMDB e US patents
SPARK	2007	*Schema graph*	Luo, Lin, Wang e Zhou	DBLP, IMDB e Mondial
BLINKS	2007	*Data graph*	He, Wang, Yang e Yu	DBLP e IMDB
Generation and Ranking of CNs	2017	*Schema graph*	Oliveira	DBLP, IMDB, Mondial e Wikipedia

Tabela 3.1: Comparativo entre os principais trabalhos.

baseados em *schema graphs*.

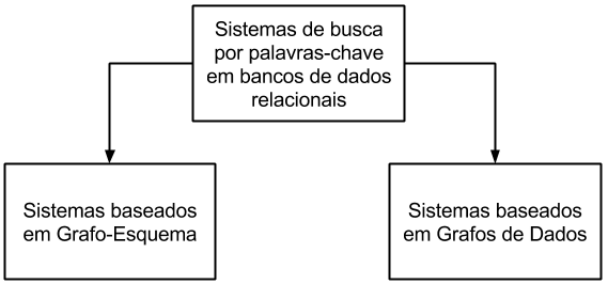


Figura 3.1: As duas categorias de sistemas de busca por palavras-chave em bancos de dados relacionais.

3.1 Arquitetura Geral

A arquitetura básica de um sistema de busca por palavras-chave em bancos de dados relacionais baseados em *schema graphs*, muito se assemelha a arquitetura de qualquer sistema de RI tradicional. A Figura 3.2 mostra o exemplo de um sistema baseado em *schema graphs*. No primeiro passo (1), o usuário insere as palavras-chave que deseja submeter ao sistema. No passo (2), o sistema procura no banco de dados pelas tuplas que contêm essas palavras-chave. Um tupla contém uma palavra-chave se qualquer um de seus atributos possuir a palavra. Após isso, o sistema busca os subconjuntos das tabelas que contêm as palavras-chave da consulta. Esses subconjuntos são chamados de **conjuntos-tupla** ou ***tuple-sets***, que serão explicados mais adiante. No passo seguinte (3), os conjuntos-tupla e o esquema do banco de dados são usados como entrada para gerar as RCs. A principal função desse passo é, dados os conjuntos-tupla, usar o esquema das tabelas do banco de dados para interconecta-los de forma que retornem resultados relevantes para o usuário.

É perceptível que muitas RCs diferentes podem ser geradas, apesar de poucas produzirem respostas relevantes. No passo (4), as RCs são avaliadas em termos de qualidade e classificadas (*ranking*) para serem apresentadas ao usuário no passo (5).

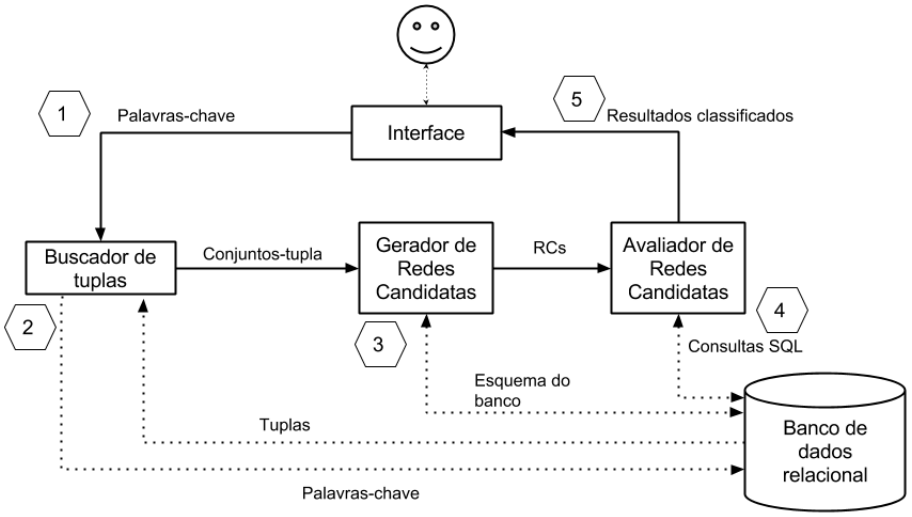


Figura 3.2: Arquitetura de um sistema de busca por palavras-chave em bancos de dados relacionais baseado em *schema graph*.

3.2 Conceitos Básicos e Terminologia

Assim como no sistema DISCOVER [21], considere como G o grafo que representa o esquema relacional do banco de dados, onde os vértices correspondem às tabelas e as arestas correspondem às restrições de integridade referencial (RIR) entre elas. Para abstrair a direção das RIR entre as tabelas, considere G_u como a versão não-direcionada do grafo G.

Para exemplificar as definições que serão colocadas adiante, será utilizado o esquema de um banco de dados utilizado por Coffman [4] conforme mostrado na Figura 3.3. Tal esquema representa um banco de dados de filmes, atores e personagens.

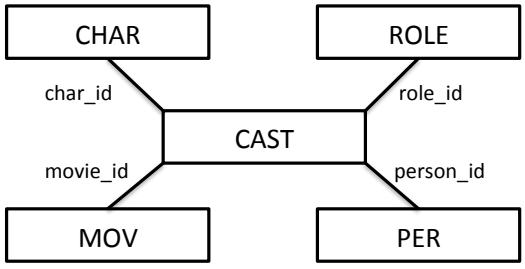


Figura 3.3: Esquema do banco de dados do IMDb utilizado por Coffman.

Definição 3.1. *Uma* **rede de junção de tuplas (RJT)** *j é uma árvore de tuplas onde para cada par de tuplas adjacentes t_i, $t_j \in j$, onde t_i e t_j são tuplas das tabelas R_i e R_j, respectivamente, existe uma aresta $\langle R_i, R_j \rangle$ em G_u e $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$.*

Considere por exemplo, uma consulta com três palavras-chave: *denzel, washington e gangster*. Uma possível RJT para este exemplo poderia ser uma tupla p_5 da tabela PER que contém a palavra *denzel* ligada a uma tupla da tabela CAST ca_3 que contém as palavras *denzel washington*, que por sua vez está conectada a uma tupla da tabela CHAR ch_1 que contém a palavra *gangster*, formando a árvore $p_5 \bowtie ca_3 \bowtie ch_1$ conforme demonstrado na Figura 3.4.

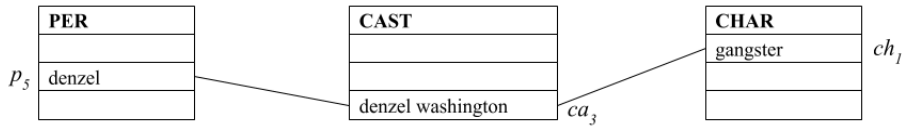


Figura 3.4: Rede de junção de tuplas formada pela consulta de exemplo *denzel, washington, gangster*.

Definição 3.2. *Dado o conjunto $Q = \{k_1, \ldots, k_n\}$ de palavras-chave, uma RJT j é uma* **rede de junção de tuplas minimal total (RJTMT)** *para Q se for* **total***, ou seja, se cada palavra-chave k_i está contida em ao menos uma tupla de j, e se for* **minimal***, ou seja, se a remoção de qualquer tupla de j faz com que a RJT não seja mais total.*

Como visto no exemplo anterior, a árvore $p_5 \bowtie ca_3 \bowtie ch_1$ não representa uma RJTMT, pois apesar de possuir todas as palavras de Q (total), não é minimal, pois se a tupla p_5 for removida, a RJT ainda permanece total. No entanto, a árvore resultante $ca_3 \bowtie ch_1$ é um exemplo válido de RJTMT conforme demonstrado na Figura 3.5.

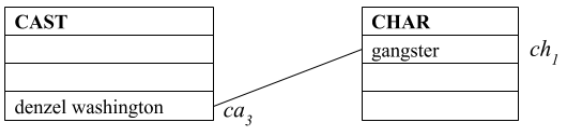


Figura 3.5: Rede de junção de tuplas minimal e total formada pela consulta de exemplo *denzel, washington, gangster*.

Definição 3.3. *Uma* **consulta por palavras-chave** *é um conjunto Q de palavras cujo resultado é o conjunto M de todas as possíveis RJTMT para as palavras em Q no conjunto das tabelas $\{R_1, \ldots, R_m\}$.*

Definição 3.4. *Seja Q o conjunto de palavras-chave da consulta e seja K um subconjunto de Q. Seja R_i uma tabela. Um* **conjunto-tupla** *de R_i sobre K é dado por*

$$R_i^K = \{t | t \in R_i \wedge \forall k \in K, k \in \mathcal{W}(t) \wedge \forall \ell \in Q - K, \ell \notin \mathcal{W}(t)\},$$

onde $\mathcal{W}(t)$ fornece o conjunto de termos (palavras) em t. Se $K = \emptyset$, o conjunto-tupla é chamado de livre*, representado por $R_i^{\{\}}$.*

Para simplificar os exemplos a seguir, será utilizada a mesma consulta do exemplo anterior: *denzel, washington, gangster*, mas representada através das iniciais *d,w,g*.

De acordo com a Definição 3.4, o conjunto-tupla (*tuple-set*) R_i^K contém as tuplas de R_i que contém todos os termos de K e nenhuma outra palavra de Q. No exemplo dado, os conjuntos-tupla que representam a RJTMT $ca_3 \bowtie ch_1$, são $\text{CAST}^{\{d,w\}}$ e $\text{CHAR}^{\{g\}}$.

Todos os possíveis conjuntos-tupla que contenham ao menos uma palavra da consulta são chamados de **não-livres**. Os conjuntos-tupla livres representam as tabelas do banco de dados que aparecerão no *schema graph*.

Definição 3.5. *Uma* **rede de junção de conjuntos-tupla** *J é uma árvore de conjuntos-tupla onde para cada par de conjuntos-tupla adjacentes R_i^K , R_j^M em J existe uma aresta $\langle R_i, R_j \rangle$ em G_u.*

No exemplo, a rede de junção de conjuntos-tupla correspondente da RJT $p_5 \bowtie ca_3 \bowtie ch_1$ seria a árvore $\text{PER}^{\{d\}} \bowtie \text{CAST}^{\{d,w\}} \bowtie \text{CHAR}^{\{g\}}$ conforme demonstrado na Figura 3.6.

Figura 3.6: Rede de junção de conjuntos-tupla formada pela consulta de exemplo *denzel, washington, gangster*.

Definição 3.6. *Dada uma consulta $Q = \{k_1, \ldots, k_n\}$, uma* **Rede Candidata** *C é uma rede de junção de conjuntos-tupla, tal que existe uma instância I do banco de dados que tem uma RJTMT $M \in C$ e nenhuma tupla $t \in M$ que mapeia para um conjunto-tupla livre $F \in C$ que contém uma ou mais palavras Q.*

De maneira simples, uma Rede Candidata (RC) é uma rede de junção de conjuntos-tupla que possui as características de uma RJTMT, ou seja, é total porque possui todas

as palavras da consulta e é minimal, pois nenhum conjunto-tupla pode ser retirado sem que ela deixe de ser total. No exemplo, a RC correspondente à RJTMT $ca_3 \bowtie ch_1$ é $\text{CAST}^{\{d,w\}} \bowtie \text{CHAR}^{\{g\}}$ conforme demonstrado na Figura 3.7.

Figura 3.7: Rede Candidata formada pela consulta de exemplo *denzel, washington, gangster*.

As RCs são em seguida transformadas em consultas estruturadas do tipo SQL para que possam ser submetidas ao banco de dados relacional.

3.3 Grafo Conjunto-Tupla

Em DISCOVER [21], a geração de RCs é um procedimento que extrai redes de junção de conjuntos-tupla do grafo que representa as possíveis formas de conectar os conjuntos-tupla encontrados no banco de dados que contém as palavras-chave da consulta. Esse grafo é chamado de *grafo conjunto-tupla* (*tuple-set graph*).

Definição 3.7. *Um grafo conjunto-tupla G_{CT} para uma consulta Q é um grafo cujos vértices são todos os conjuntos-tupla não-livres $R_i^{K_i}$, onde $K_i \subseteq Q$, incluindo conjuntos-tupla livres, e existe uma aresta $\langle R_i^{K_i}, R_j^{K_j} \rangle$ em G_{CT} se o grafo do esquema G_u possuir uma aresta $\langle R_i, R_j \rangle$.*

A Figura 3.8 representa o grafo G_{CT} que demonstra a consulta de exemplo *denzel, washington, gangster* para o mesmo banco de dados.

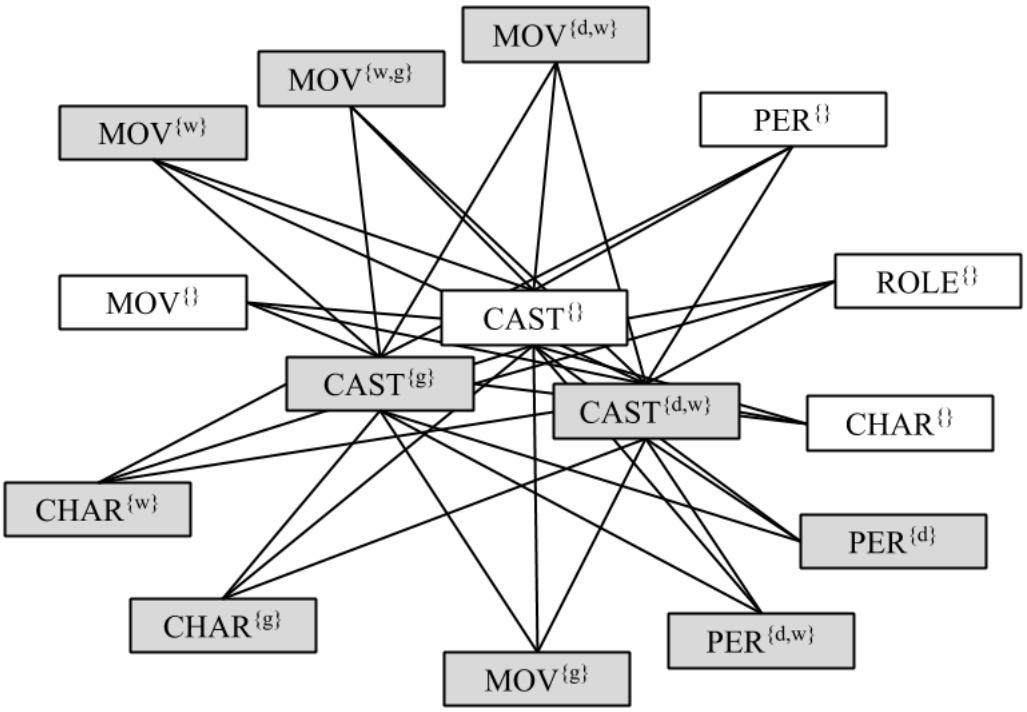


Figura 3.8: Grafo G_{CT} para a consulta *denzel, washington, gangster* no banco de dados IMDb.

No entanto, nem toda árvore retirada do grafo G_{CT} pode ser considerada uma RC. Ainda de acordo com DISCOVER [21], existem duas restrições a serem observadas quanto à estrutura de uma dada solução para que essa seja considerada válida. A primeira diz que:

Definição 3.8. *Uma rede de junção de tuplas J não é minimal se possui uma tupla sem palavras-chave como folha. Essa condição portanto não permite conjuntos-tupla livres como folhas.*

A segunda restrição é chamada de *Condição de Poda* (*Pruning Condition*) e é dada conforme segue:

Definição 3.9. *Uma Rede Candidata não contém uma subárvore na forma de* $R^K - S^L - R^M$*, onde R e S são tabelas e o grafo do esquema G possui uma aresta* $R \rightarrow S$*.*

Dessa forma, a partir do grafo G_{CT}, é possível obter as RCs ao extrair árvores que possuam as três seguintes características:

- O conjunto de conjuntos-tupla presentes na árvore deve formar uma cobertura total e minimal com as palavras-chave da consulta.

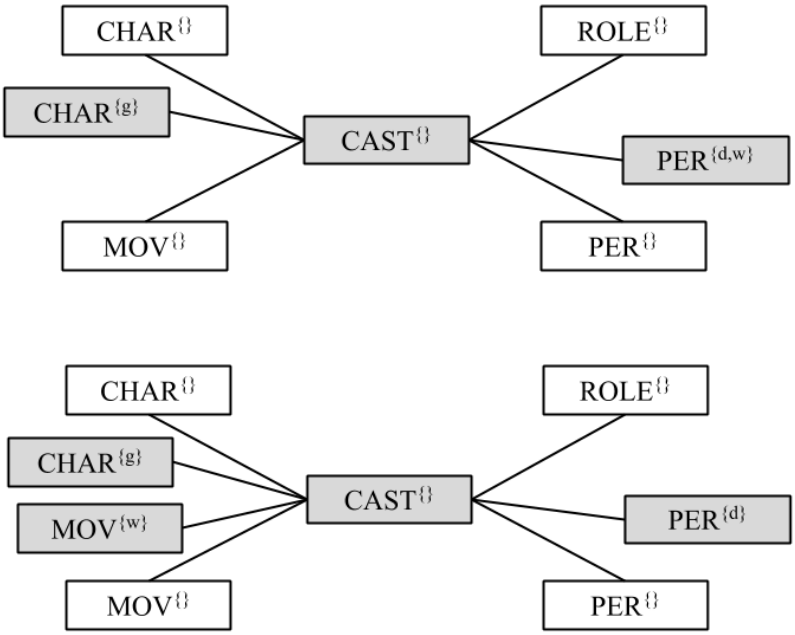


Figura 3.9: Duas árvores retiradas de G_{CT} que também são RCs.

- Não podem existir conjuntos-tupla livres como folhas na árvore.
- A árvore não pode possuir a estrutura definida pela Condição de Poda.

A Figura 3.9 apresenta duas árvores marcadas em cinza retiradas de G_{CT} que são RCs válidas e a Figura 3.10 (a) ilustra um subgrafo de G_{CT} cujos conjuntos-tupla formam uma cobertura total e minimal com as palavras de Q, mas neste caso, existem quatro diferente formas de interconectar estes conjuntos-tupla. Nesse caso em particular, todas as quatro árvores entram na Condição de Poda, e portanto não são RCs.

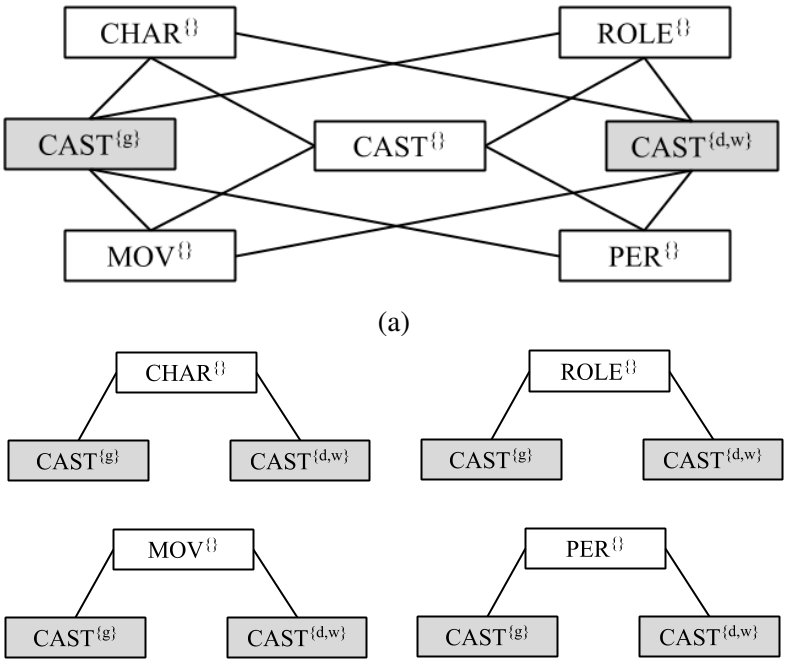


Figura 3.10: Subgrafo de G_{CT} (a) e quatro árvores derivadas (b).

Capítulo 4

O Problema da Geração de Redes Candidatas

Neste capítulo será apresentada uma modelagem teórica para problema da geração de Redes Candidatas (do inglês, *Candidate Network Generation Problem* - CNGP), evidenciando-se aspectos teóricos da aplicação que resulta em uma proposta de modelagem em grafos considerando uma variação do problema da árvore de Steiner combinada com o problema de coloração de arestas em grafos, de forma que atenda às restrições adicionais impostas pela aplicação em banco de dados. Esee problema, então, está sendo denominado de problema da árvore de Steiner com arestas coloridas (do inglês, *Steiner Tree with Colored Edges Problem* - STCEP).

4.1 Modelagem Teórica do Problema

Seja G_{CT} o grafo conjunto-tupla formado para uma dada consulta de palavras-chave Q, a solução esperada para o problema da geração de RCs, consiste em subgrafos de G_{CT} que sejam árvores e que contenham conjuntos-tupla não-livres que formem uma cobertura total e minimal com as palavras-chave da consulta, conforme já mostrado no Capítulo 3.

No problema da árvore de Steiner em grafos, dado um grafo qualquer G, a solução deste problema também consiste em um subgrafo T de G, onde T é uma árvore que contém os vértices de um subconjunto chamado de **terminais**. Como já explicado na Definição 2.4, uma árvore é um grafo simples, conexo e acíclico. Na versão de otimização do problema, também chamado de árvore de Steiner mínima, busca-se uma árvore T que

conecte os terminais em G de forma que a soma dos pesos das arestas escolhidas seja mínima.

O problema da árvore de Steiner e o problema da geração de RCs se relacionam de forma direta, tendo ambos como entrada um grafo e um subconjunto de vértices, e como saída uma árvore do grafo original que conecta os vértices desse subconjunto. Entretanto, no problema da geração de RCs, o conjunto de terminais não é fixo, pois como já visto anteriormente, é possível formar mais de um conjunto de terminais que representem coberturas totais e minimais com as palavras-chave da consulta. Isso implica que é possível gerar mais de uma RC a partir do grafo conjunto-tupla, além do fato de que pode haver mais de uma árvore de Steiner de custo mínimo (com o mesmo conjunto de terminais).

A seguir serão apresentadas definições adicionais para mapear de forma mais clara os aspectos que definem tais subconjuntos de vértices a serem conectados. As seguintes definições baseiam-se nos conceitos prévios introduzidos no Capítulo 3.

Definição 4.1. *Seja Q o conjunto de palavras-chave da consulta, um* conjunto de termos *K é um conjunto tal que $K \subseteq Q$ e $K \neq \emptyset$.*

Um **conjunto de termos** portanto, representa um possível subconjunto não-vazio das palavras-chave da consulta. A partir da Definição 3.4, é possível dizer que um conjunto-tupla R_i^K representa o subconjunto de tuplas de R_i onde cada tupla contêm o conjunto de termos K e nenhum outro termo de Q.

Definição 4.2. *Seja Q o conjunto de palavras-chave da consulta e I uma instância do banco de dados. $\mathcal{R}_Q$ é o conjunto formado por todos os conjuntos-tupla não-livres $R_i^K \in I$, onde K é um conjunto de termos de Q.*

O conjunto $\mathcal{R}_Q$ possui todos os vértices presentes no grafo G_{CT}, exceto os conjuntos-tupla livres. Conforme a Definição 3.6, os conjuntos de termos dos conjuntos-tupla presentes em uma RC precisam formar uma cobertura total e minimal com as palavras de Q. Por conveniência, o subconjunto M de conjuntos-tupla que formam tal cobertura será chamado de **match**.

Definição 4.3. *Seja Q o conjunto de palavras-chave da consulta e $\mathcal{R}_Q$ o conjunto de conjuntos-tupla encontrados em uma instância I do banco de dados. Seja $M \subset \mathcal{R}_Q$ tal que $M = \{R_1^{K_1}, \ldots, R_m^{K_m}\}$, onde cada K_i é um conjunto de termos de Q. Dizemos que M é um **match** para Q se $K_1 \cup \ldots \cup K_m = Q$ e $(K_1 \cup \ldots \cup K_m) \setminus K_i \neq Q$ para qualquer K_i.*

Por outro lado, nem toda árvore no grafo conjunto-tupla G_{CT} contendo um *match* pode ser considerada uma RC, pois a Condição de Poda colocada na Definição 3.9 atesta que tais árvores não podem conter subárvores com a estrutura descrita pela condição.

Para exemplificar as definições e conceitos apresentados, será reutilizada a consulta de exemplo do Capítulo 3 com o grafo G_{CT} reapresentado na Figura 4.1. Os conjuntos-tupla não-livres estão marcados em cinza.

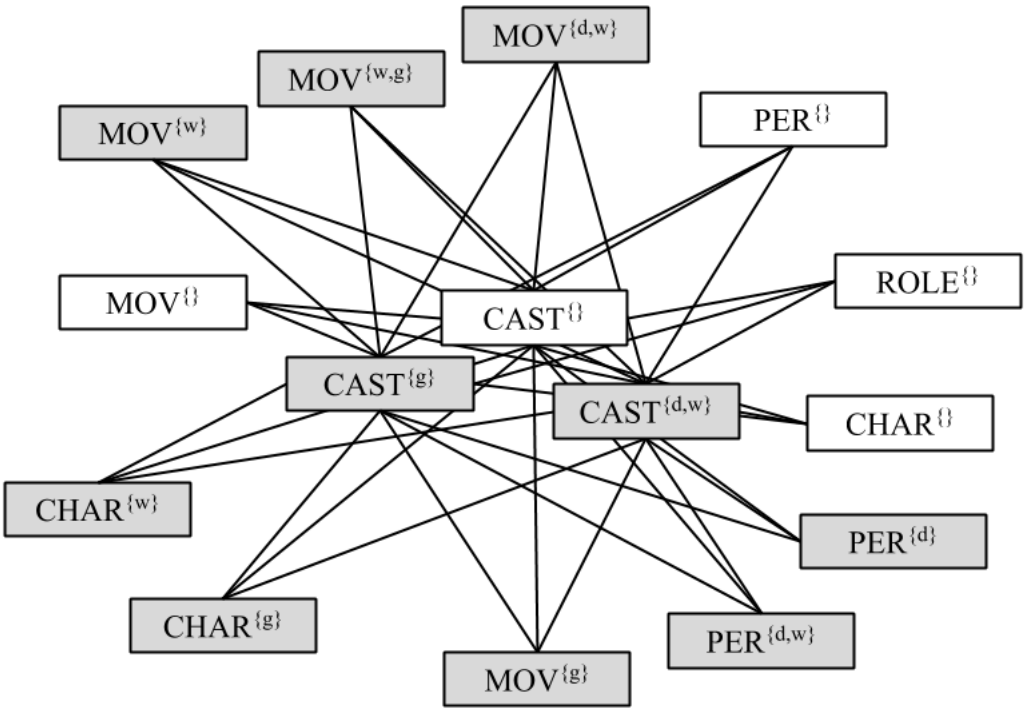


Figura 4.1: Grafo conjunto-tupla G_{CT} para a consulta *denzel, washington, gangster* no banco de dados IMDb.

Na Tabela 4.1 têm-se à esquerda duas possíveis coberturas totais e minimais a partir dos conjuntos de termos dos conjuntos-tupla em $\mathcal{R}_Q$ e ao lado todos os respectivos *matches* de cada uma.

Na Figura 4.2 são apresentadas algumas possíveis RCs para o *match* $M = \{\text{CAST}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$. Note que os conjuntos-tupla livres fazem a conexão entre os não-livres e que os não-livres são sempre folhas.

Cobertura total e minimal	Todos os *matches*
{{d,w} {g}}	$M_1 = \{\text{CAST}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$ $M_2 = \{\text{MOV}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$ $M_3 = \{\text{MOV}^{\{d,w\}}, \text{CHAR}^{\{g\}}\}$ $M_4 = \{\text{CAST}^{\{d,w\}}, \text{CAST}^{\{g\}}\}$ $M_5 = \{\text{CAST}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$ $M_6 = \{\text{CAST}^{\{d,w\}}, \text{CHAR}^{\{g\}}\}$ $M_7 = \{\text{PER}^{\{d,w\}}, \text{CAST}^{\{g\}}\}$ $M_8 = \{\text{PER}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$ $M_9 = \{\text{PER}^{\{d,w\}}, \text{CHAR}^{\{g\}}\}$
{{d} {w,g}}	$M_1 = \{\text{PER}^{\{d\}}, \text{MOV}^{\{w,g\}}\}$

Tabela 4.1: Exemplo de todos os *matches* para duas possíveis coberturas totais e minimais.

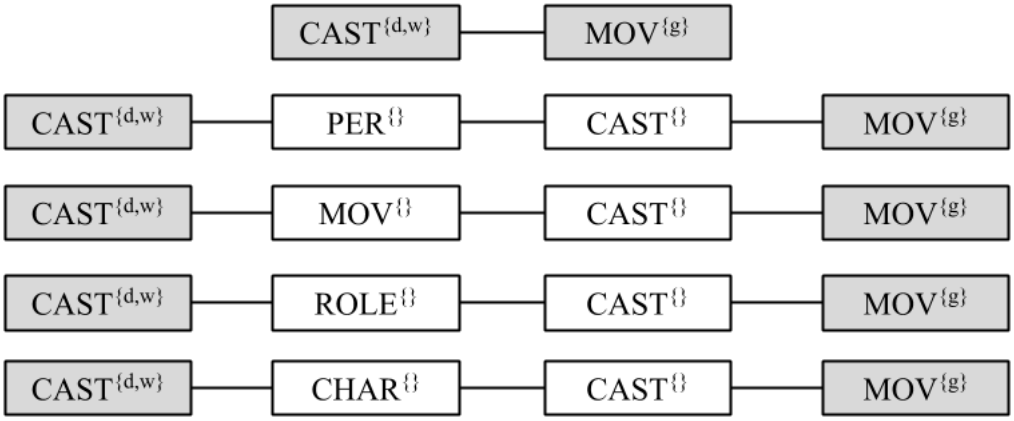


Figura 4.2: Cinco possíveis RCs para o *match* $M = \{\text{CAST}^{\{d,w\}}, \text{MOV}^{\{g\}}\}$.

4.2 Modelagem do Problema em Grafos

De modo a representar todas as características e restrições envolvidas no problema de geração de redes candidatas (CNGP), foi observado que não seria suficiente modelar teoricamente como um problema de árvore de Steiner em grafos, pois alguns casos ocorridos na estrutura de solução do CNGP não se enquadravam como apresentado anteriormente. Sendo assim, trabalhou-se em uma variação do problema da árvore de Steiner em grafos e como nenhuma das existentes na literatura contemplava o desejado, propõe-se uma nova variação fruto da combinação com o problema da coloração de arestas em grafos. Esse último problema clássico de teoria dos grafos está descrito a seguir (Problema 4.1), em sua versão de decisão.

PROBLEMA DA COLORAÇÃO DE ARESTAS EM GRAFOS
Instância: Um grafo $G = (V, E)$ e uma função $c : E \rightarrow \{1, \ldots, \psi\}$.
Pergunta: É possível atribuir ψ cores para as arestas em E tal que se $v \neq w$ e $(u, v), (u, w) \in E$ então $c(u, v) \neq c(u, w)$?

Problema 4.1: Instância de entrada e pergunta para o problema da coloração de arestas em grafos.

No problema da coloração de arestas [19], o objetivo é obter uma coloração para as arestas do grafo G utilizando no máximo ψ cores de forma que não existam arestas adjacentes da mesma cor. A Figura 4.3 apresenta um exemplo de um grafo com tal coloração.

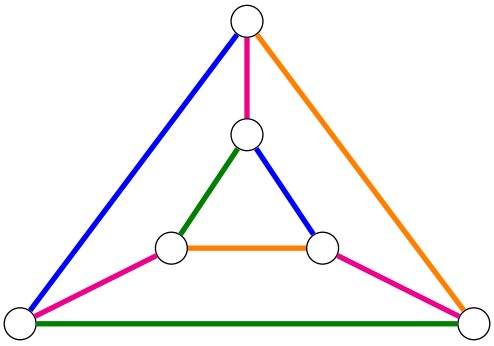


Figura 4.3: Exemplo de grafo com coloração de arestas.

Em sua versão de otimização, busca-se o valor mínimo necessário de cores para se conseguir a coloração das arestas do grafo. Esse valor é chamado de **índice cromático** que é representado por $\psi(G)$. A Figura 4.4 apresenta o mesmo grafo da Figura 4.3 mas com o mínimo de cores necessário para colorir suas arestas, que no caso é 3.

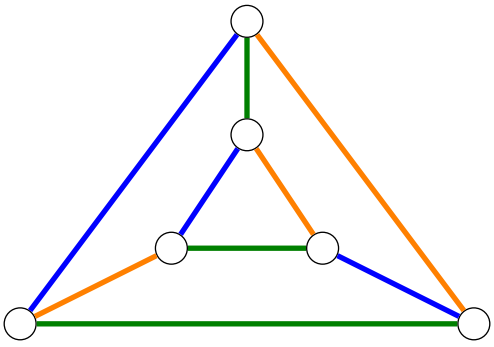


Figura 4.4: Exemplo de coloração de arestas utilizando o mínimo possível de cores.

O problema da coloração de arestas em grafos, assim como o problema da árvore de Steiner, também já foi provado ser NP-Completo [19].

A seguir é apresentada a descrição de uma variação do problema da árvore de Steiner

com coloração de arestas (Problema 4.2), que modela em plenitude o problema da geração de Redes Candidatas em banco de dados relacionais (CNGP). Esse problema representa uma nova proposta para o problema da árvore de Steiner em grafos: o problema da árvore de Steiner com coloração de arestas (do inglês, *Steiner Tree with Colored Edges Problem* - STCEP)

PROBLEMA DA ÁRVORE DE STEINER COM ARESTAS COLORIDAS (STCEP)
Instância: Um grafo ponderado $G = (V, E)$, um conjunto de vértices $W \subset V$, uma função $c : E \rightarrow \{1, \ldots, \psi\}$ e um inteiro x.
Pergunta: Existe uma árvore T de G, tal que $W \subseteq V(T)$, $\|E(T)\| \leq x$ e que possua uma coloração de arestas válida?

Problema 4.2: Instância de entrada e pergunta para o problema da árvore de Steiner com coloração de arestas em grafos.

O problema possui a mesma estrutura do problema da árvore de Steiner em grafos, a diferença é que nem toda árvore de Steiner é solução válida para este novo problema. Na instância do grafo de entrada já possui uma função de coloração para suas arestas, e, deste modo, uma árvore só é considerada uma solução válida para o STCEP se apresentar uma coloração onde não hajam arestas adjacentes de mesma cor.

A Figura 4.5 mostra um exemplo de um grafo com coloração de arestas e os terminais marcados em vermelho, ao lado uma árvore de Steiner com coloração inválida e logo em seguida uma árvore de Steiner com coloração válida (árvore de Steiner com arestas coloridas)

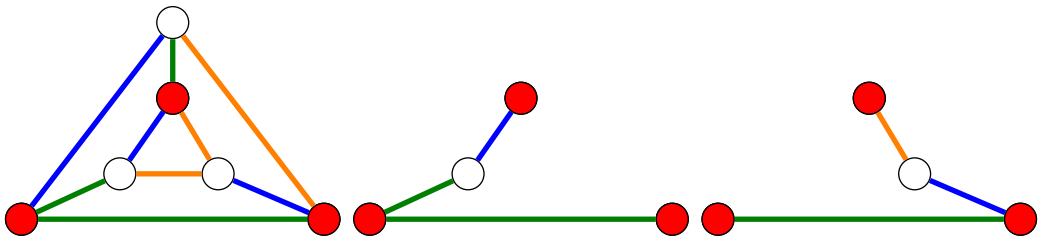
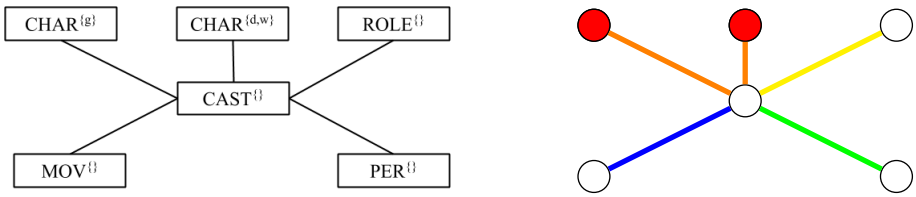


Figura 4.5: (a) Grafo com arestas coloridas e terminais. (b) Árvore de Steiner com arestas coloridas inválida. (c) Árvore de Steiner com arestas coloridas válida.

Esta variação do problema permite uma modelagem em grafos para o problema da geração de RCs com o seguinte mapeamento de parâmetros de um para o outro:

- O grafo G_{CT} é modelado como o grafo de entrada G.
- Um dado *match* M é modelado como o conjunto de terminais W.
- O peso das arestas é considerado sempre igual a 1.
- É atribuída uma cor única para cada aresta de G, exceto na condição de poda R^K – S^L – R^M que é representada com as arestas R^K – S^L e R^M – S^L possuindo cores iguais em G.

A Figura 4.6 mostra uma instância do problema da geração de RCs (CNGP) modelada em grafos para o STCEP. Note que para cada aresta do grafo G é atribuída uma cor única, exceto para o par de arestas $CHAR^{\{g\}}$ – $CAST^{\{\}}$ e $CHAR^{\{d,w\}}$ – $CAST^{\{\}}$ (condição de poda) cujas arestas ficam com a mesma cor.



(a) (b)

Figura 4.6: (a) Grafo G_{TS} de entrada do CNGP. (b) Grafo remodelado para STCEP.

A partir das definições apresentadas, têm-se o *Lema* 1 a seguir.

Lema 1. *Seja Q uma consulta, toda Rede Candidata é uma árvore de Steiner no grafo* G_{CT} *cujos terminais são vértices de um possível match para Q.*

Prova: Uma RC é sempre uma árvore, pois por definição uma rede de junção de conjuntos-tupla é sempre uma árvore de acordo com a definição 3.5. Também pela definição 3.6, os conjuntos-tupla de uma RC sempre formam uma cobertura total e minimal com as palavras da consulta, ou seja formam um *match*, e todas as folhas de uma RC devem ser sempre conjuntos-tupla não-livres conforme a Definição 3.8. Portanto uma RC sempre é uma árvore que interliga os vértices de um *match*, e cujas folhas são sempre vértices desse *match*.

Assim têm-se o *Lema* 2.

Lema 2. *Dado um match M para uma consulta Q, toda rede de junção de conjuntos-tupla para M, que satisfaz a Condição de Poda e cujas folhas pertencem a M, é uma Rede Candidata.*

Prova: Por ser uma rede de junção de conjuntos-tupla satisfaz a estrutura de árvore necessária para ser uma RC. Ao conectar os elementos de um *match* satisfaz a cobertura obrigatória de conjuntos-tupla. Sejam as folhas pertencentes ao *match*, garante que todas as folhas sejam conjuntos-tupla não-livres. E por fim satisfaz a Condição de Poda. Portanto o lema 2 é verdadeiro.

A partir das definições apresentadas, segue o Teorema 1,

Teorema 1. *Seja G_{CT} o grafo conjunto-tupla para o conjunto Q de palavras-chave, e seja $M \subset \mathcal{R}_Q$ um match para o conjunto Q, uma Rede Candidata C é uma árvore de Steiner com coloração de arestas em G_{CT}, onde M representa o conjunto de terminais.*

Prova: Pelo Lema 1 foi provado que em uma RC, os vértices são interconectados como terminais de forma semelhante ao problema da árvore de Steiner em grafos. Pelo Lema 2 no entanto, somente são RC as árvores que respeitam a Condição de Poda (coloração de arestas diferentes), portanto o Teorema 1 é verdadeiro.

A partir do Teorema 1, é possível especificar a versão de decisão do problema da geração de RCs a partir de sua instância de entrada e da pergunta que deve ser respondida com SIM ou NÃO conforme definido no Problema 4.3.

PROBLEMA DA GERAÇÃO DE REDES CANDIDATAS (CNGP)
Instância: Um *schema graph G*, um conjunto Q de palavras-chave, um conjunto $\mathcal{R}_Q$ de conjuntos-tupla e um grafo conjunto-tupla G_{CT}.
Pergunta: Existe uma ou mais Redes Candidatas $C \in G_{CT}$?

Problema 4.3: Instância de entrada e pergunta para o problema da geração de RCs.

Capítulo 5

Aspectos de Complexidade Computacional

Neste capítulo são apresentados aspectos de complexidade computacional do problema abordado, tanto em relação ao caso teórico geral, quanto considerando questões da aplicação prática. Sendo assim, será apresentada a prova de NP-Completude para o problema da árvore de Steiner com arestas coloridas (do inglês, *Steiner Tree with Colored Edges Problem* (STCEP)). Em seguida, são apresentadas análises de complexidade no pior caso (algoritmo geral) e em questões da aplicação prática, principalmente considerando que o número de palavras-chave na busca geralmente é fixo e pequeno, no problema da geração de Redes Candidatas (RCs).

5.1 Prova de NP-Completude para o problema STCEP

Para provar que o STCEP é NP-Completo, será inicialmente provado que o mesmo está em NP, e posteriormente que também está em NP-difícil através de uma transformação em tempo polinomial para outro problema NP-Completo conhecido.

5.1.1 STCEP pertence à Classe de Problemas NP

Um problema qualquer pertence à classe NP de problemas quando é possível provar que existe um algoritmo polinomial capaz de reconhecer se uma dada solução é válida para o problema [14].

Lema 3. *STCEP* $\in$ *NP*

Para provar que uma solução do problema STCEP é válida, basta percorrer os vértices utilizando uma busca em profundidade. Caso todos os terminais sejam visitados e não haja vértices que compartilhem arestas de mesma cor, a solução portanto é válida. O Algoritmo 1 apresenta o pseudo-código para realizar tal verificação e responde SIM ou NÃO se a instância é válida.

Algoritmo 1: Reconhecimento de solução para o STCEP

```
Entrada: Árvore T, Conjunto de vértices W, Função de cores c
Saída: SIM ou NÃO
para cada vértice u ∈ T faça
    se u ∈ W então
        W ← W \ u
    aux ← []
    para cada vértice v adjacente a u faça
        aux[c(u,v)]++
        se aux[c(u,v)] > 1 então
            responde NÃO
se W = ∅ então
    responde SIM
senão
    responde NÃO
```

É possível percorrer todos os vértices em tempo $O(n)$ e analisar se o vértice é um terminal em tempo $O(k)$, onde k representa o número de terminais a serem conectados. Portanto o algoritmo tem desempenho polinomial em tempo $O(kn)$ e portanto STCEP pertence à classe de problemas NP.

5.1.2 STCEP pertence à Classe de Problemas NP-difíceis

Para fazer esta demonstração, será utilizado o problema do *Exact Cover by 3-Sets* (X3C) que é descrito no Problema 5.1.

EXACT COVER BY 3-SETS (X3C)

Instância: um conjunto finito X com $|X| = 3q$ e uma coleção C de subconjuntos de 3-elementos de X, $C = \{C_1, \ldots, C_n\}$, $C_i \subseteq X$, $|C_i| = 3, 1 \leq i \leq n$.

Pergunta: C contém um *exact cover* para X, ou seja, uma subcoleção $C' \subseteq C$ tal que todo elemento X ocorre em exatamente um membro de C'?

Problema 5.1: Instância de entrada e pergunta para o problema exact cover by 3-sets.

A partir de C' é possível perceber que os membros de C' formam uma partição com os membros de X.

Lema 4. *STCEP* $\in$ *NP-Difícil.*

Considere como instância do X3C os conjuntos $X = \{x_1, \ldots, x_{3q}\}$ e uma coleção de conjuntos de 3-elementos $C = \{C_1, \ldots, C_n\}$. É necessário construir uma árvore de Steiner colorida em grafos especificando o grafo $G = (V, E)$, o conjunto de terminais W, uma função de coloração para as arestas c, e o inteiro x.

- o conjunto de vértices fica definido como:

$$V(G) = \{v\} \cup \{c_1, \ldots, c_n\} \cup \{x_1, \ldots, x_{3q}\} \tag{5.1}$$

  é inserido um vértice adicional v, um vértice para cada conjunto em C e um vértice para cada elemento de X.

- o conjunto de arestas fica definido como:

$$E(G) = \{vc_1, \ldots, vc_n\} \cup (\bigcup_{x_j \in C_i} c_i x_j) \tag{5.2}$$

  existe uma aresta de v para cada vértice c_i, e uma aresta $c_i x_j$ se o elemento x_j aparece no conjunto C_i da instância.

- o conjunto de terminais fica definido como:

$$W = \{v, x_1, \ldots, x_{3q}\} \tag{5.3}$$

- a função de coloração das arestas fica definida como:

$$c : E \to \{1, \ldots, |E|\} \tag{5.4}$$

  no caso para cada aresta é atribuída uma cor única.

- seja $x = 4q$.

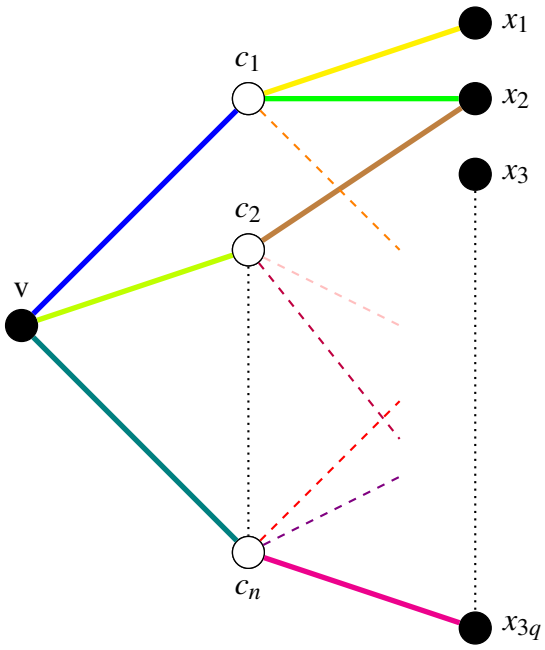


Figura 5.1: Grafo criado a partir de instância do X3C.

A Figura 5.1 mostra o grafo construído no processo de transformação. Os vértices pintados de preto representam os terminais.

Afirmação 1: Se existe um *exact 3-cover* na instância X, então existe uma árvore de Steiner com arestas coloridas na instância Y.

Supondo que existe um *exact 3-cover* C' para o problema X3C, este deve ter q subconjuntos (assuma $\{C_1,\ldots,C_q\}$), então a árvore contendo as arestas:

- $vc_1,\ldots,vc_q$
- c_ix_j, se $x_j \in C_i$ e $1 \leq i \leq q$

é uma árvore de Steiner que resolve o problema com custo igual a $q+3q = 4q = x$.

Afirmação 2: Se existe uma árvore de Steiner com arestas coloridas transformada Y, então, existe um *exact 3-cover* na instância X.

Supondo que existe uma árvore de Steiner com arestas coloridas com custo máximo de $4q$. Dado que é uma árvore, possui no máximo $4q+1$ arestas. Pela definição, a árvore deve interconectar todos os terminais $(v,x_1,\ldots,x_q)$, portanto passa por no máximo q vértices do tipo c. Como o grau desses vértices é 3, não é possível chegar a todos os terminais se a árvore tiver menos que $4q+1$ arestas de peso igual a 1. Portanto é possível concluir que a solução possui exatamente q vértices $\{c_1,\ldots,c_q\}$ que corresponde a uma solução para X3C dada por $\{C_1,\ldots,C_q\}$.

Teorema 2. *STCEP pertence à classe de problemas NP-Completos.*

Prova: Pelo *Lema* 3 foi provado que STCEP $\in$ NP e pelo *Lema* 4 foi provado que X3C $\Leftrightarrow$ STCEP, desta forma, STCEP $\in$ NP-Completo.

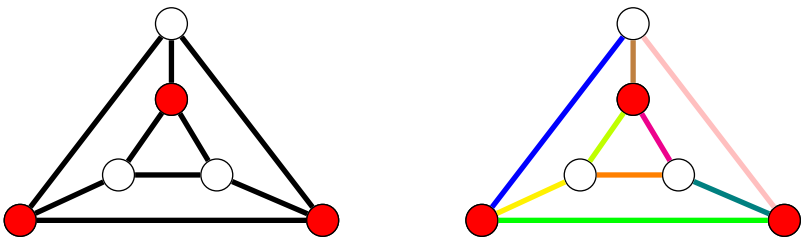
Uma forma mais simples para demonstrar que o STCEP pertence à classe de problemas NP-Completos, é fazer uma redução a partir do problema original da árvore de Steiner em grafos, que já foi provado pertencer ao conjunto dos problemas NP-Completos [25].

Lema 5. *O problema da árvore de Steiner em grafos é NP-Completo.*

A redução do problema da árvore de Steiner para o STCEP é trivial, pois os problemas possuem quase os mesmos parâmetros de entrada: um grafo conexo G, um subconjunto de terminais $W \subseteq V(G)$, e um inteiro x. A diferença é que o STCEP apresenta como parâmetro adicional a função c de coloração das arestas . Para realizar a redução, basta que essa função designe uma cor diferente para cada aresta do grafo original G para o STCEP.

$$c : E \rightarrow \{1, \ldots, |E|\} \tag{5.5}$$

A Figura 5.2 mostra um grafo G de exemplo com seu respectivo conjunto de terminais W marcados em vermelho e ao lado a mesma instância reduzida para o STCEP com uma cor única para cada aresta.



(a) (b)

Figura 5.2: (a) Exemplo de grafo com terminais em vermelho e (b) Instância equivalente com função de coloração para as arestas.

Teorema 3. *O problema da árvore de Steiner com coloração de arestas pode ser provado NP-Completo através de uma redução polinomial para o problema da árvore de Steiner em grafos.*

5.2 Limites Assintóticos para o Cenário de Aplicação

A seguir serão apresentadas as análises para os cenários teóricos e práticos de aplicação do CNGP.

5.2.1 Análise do Cenário Teórico

Para a análise do cenário teórico, é necessário considerar as variáveis que influenciam na complexidade computacional do algoritmo proposto para resolver o problema da geração de RCs, que no caso são o número de tabelas do banco de dados (R), o número de restrições de integridade referencial entre essas tabelas (m), o número de palavras-chave na consulta ($|Q|$) e o conjunto de conjuntos-tupla ($\mathcal{R}_Q$) encontrados nas tabelas do banco de dados .

Dentre tais variáveis, a que mais influencia a dificuldade computacional de uma instância qualquer é a distribuição das palavras nas tabelas ($\mathcal{R}_Q$). Portanto, para análise do cenário teórico, será considerado o cenário em que todas as $|Q|$ palavras-chave estão presentes em todas as tabelas do banco, e também que em cada tabela, estão presentes todos os possíveis conjuntos-tupla.

O passo intermediário de encontrar os possíveis *matches* pode ser caracterizado como o problema de cobertura minimal de conjuntos finitos [18]. É possível mostrar que qualquer cobertura minimal de um conjunto de n elementos, tem no máximo n subconjuntos [18], sendo assim, para cada subconjunto de conjuntos de termos ser uma cobertura mínima, tem de haver no máximo $|Q|$ elementos, caso contrário, seria possível remover qualquer um de seus elementos e ainda assim manter a propriedade da cobertura.

Entretanto, é preciso considerar todas as possíveis coberturas com 1 até $|Q|$ conjuntos de termos, contanto que sejam totais e minimais. Por exemplo, considere as palavras d, w, g; em uma só tabela é possível encontrar os seguintes conjuntos de termos no pior caso: $\{d\}, \{w\}, \{g\}, \{d, w\}, \{d, g\}, \{w, g\}$, $\{d, w, g\}$.

Nesse cenário, como todos os possíveis conjuntos-tupla estão presentes em todas as tabelas, uma fórmula para cálculo de todos os possíveis *matches* é através da Equação 5.6 de Hearne e Wagner [18]. Seja $\mu(n, k)$ o número de coberturas minimais de $\{1, \ldots, n\}$

com k elementos e seja S o número de Stirling.

$$\mu(n,k) = \frac{1}{k!} \sum_{m=k}^{min(n,2^k-1)} \binom{2^k-k-1}{m-k} m!S(n,m) \tag{5.6}$$

No entanto, para cada *match*, todas as possíveis combinações de tabelas do banco de dados podem ser utilizadas para construir o mesmo *match*. Por exemplo, considere que o banco de dados possui 3 tabelas, $R = (X,W,Y)$, é possível formar o subconjunto total e minimal $\{\{d\},\{w,g\}\}$ com 9 (3^2) combinações de tabelas, resultando em 9 *matches*:

$$X^{\{d\}},X^{\{w,g\}}$$
$$X^{\{d\}},W^{\{w,g\}}$$
$$X^{\{d\}},Y^{\{w,g\}}$$
$$Y^{\{d\}},X^{\{w,g\}}$$
$$Y^{\{d\}},W^{\{w,g\}}$$
$$Y^{\{d\}},Y^{\{w,g\}}$$
$$W^{\{d\}},X^{\{w,g\}}$$
$$W^{\{d\}},W^{\{w,g\}}$$
$$W^{\{d\}},Y^{\{w,g\}}$$

A Equação 5.7 representa o cálculo para o número total de *matches*.

$$\sum_{i=1}^{|Q|} R^i \mu(|Q|,i) \tag{5.7}$$

Em seguida é necessário encontrar todas as possíveis árvores de Steiner no grafo G_{CT} considerando os conjuntos-tupla de cada *match* como terminais. O trabalho de Dourado, Oliveira e Protti [8], apresenta uma análise acerca do problema de árvores de Steiner útil para analisar os limites assintóticos.

Os autores mostram que em uma árvore de Steiner, existem vértices chamados de **roteadores**, e que em toda árvore de Steiner, o caminho entre dois terminais é sempre o caminho mínimo, bem como também é um caminho mínimo entre dois roteadores como também entre um terminal e um roteador. O número máximo de roteadores necessários é sempre $k-2$, onde k é o número de terminais do grafo, e o número mínimo é zero. Isso é fácil de verificar, pois quando $k = 2$, a árvore de Steiner resultante necessária para interconectar os vértices terminais é sempre um caminho mínimo.

No trabalho de Dourado, Oliveira e Protti [8], é apresentado um algoritmo para enu-

meração de todas as árvores mínimas de Steiner, e para isso considera-se todas as possíveis árvores com $0, ..., k-2$ roteadores. É possível utilizar esse algoritmo considerando o grafo G_{CT} com as tabelas R como os vértices e os *tuple-sets* de cada subconjunto como o conjunto de terminais a serem conectados em G_{CT}, e assim é possível definir o limite assintótico para o cenário teórico utilizando-se o algoritmo proposto para um dado *match* baseado na complexidade do algoritmo onde α representa o número de árvores de Steiner enumeradas e m o número de arestas no grafo. A Equação 5.8 mostra a complexidade de se obter as árvores de Steiner para um dado *match* e a Equação 5.9 mostra a complexidade total no pior caso para obter as árvores de Steiner de todos os *matches*.

$$O(R^2(R+m) + R^{|Q|-2} + R\alpha) \tag{5.8}$$

$$O(\sum_{i=1}^{|Q|} R^i \mu(|Q|, i)(R^2(R+m) + R^{|Q|-2} + R\alpha)) \tag{5.9}$$

5.2.2 Análise do Cenário Prático

Conforme demonstrado na seção anterior, fica claro que a complexidade computacional esperada para o pior caso no cenário teórico para o algoritmo proposto para o problema da geração de RCs é de ordem exponencial para os parâmetros do problema. No entanto, no pior caso estimado, todas as palavras da consulta aparecem em todas as tabelas em todas as combinações possíveis.

De maneira empírica é possível afirmar que o pior caso estimado é um cenário altamente improvável. O ideal seria obter um comportamento médio da ocorrência e distribuição das palavras nas tabelas do banco de dados, mas isso não é trivial pois cada base de dados possui uma distribuição única que não serve como base para uma generalização.

No entanto, com base nos principais bancos de dados de teste disponíveis como *benchmark* [4], vamos assumir que a maioria das palavras de um banco de dados ocorre em 1 ou 2 tabelas do banco. Sendo assim é possível reescrever a equação de complexidade para o algoritmo proposto para a geração de RCs considerando tais dados a partir da Equação 5.7.

Considere uma consulta de exemplo com $Q = \{a, b, c\}$, que cada palavra apareça nas mesmas duas tabelas do banco de dados (X e Y) e que assim também formem todos os

possíveis conjuntos de termos conforme abaixo:

$$\mathcal{R}_Q = \{X^{\{a\}}, X^{\{b\}}, X^{\{c\}}, X^{\{a,b\}}, X^{\{a,c\}}, X^{\{b,c\}}, X^{\{a,b,c\}}, Y^{\{a\}}, Y^{\{b\}}, Y^{\{c\}}, Y^{\{a,b\}}, Y^{\{a,c\}}, Y^{\{b,c\}}, Y^{\{a,b,c\}}\}.$$

Dessa forma é possível recalcular a Equação 5.7:

$$\sum_{i=1}^{|Q|} R^i \mu(|Q|, i) = \sum_{i=1}^{3} 2^i \mu(3, i) = 2 * 1 + 4 * 6 + 8 * 1 = 34 \tag{5.10}$$

É possível perceber que o número de *matches* é quase uma constante quando o número de palavras-chave da consulta é pequeno. Aplicando na Equação 5.9, considere que $m \approx R$ pois os grafos do esquema de bancos de dados não possuem muitas conexões entre as tabelas e considere que $\alpha = 1$ para simplificar. É possível perceber que o fator mais influente na complexidade computacional do algoritmo proposto para o problema da geração de RCs é o número de palavras-chave e o número de tabelas, mas que para até 5 palavras-chave, somente o número de tabelas no banco de dados influencia na complexidade do algoritmo. Nesse caso o algoritmo possui custo polinomial apesar de ser exponencial.

$$O(34 * (R^2(R+R) + R^{|Q|-2} + R)) = O(R^3 + R^{|Q|-2}) \tag{5.11}$$

Capítulo 6

Algoritmos, Experimentos Computacionais e Análise Empírica

Neste capítulo são apresentados os algoritmos investigados e implementados para o problema da árvore de Steiner em grafos, que usa também algoritmos para o problema da geração de todas as árvores geradoras mínimas, que foram assim também investigados e implementados. Experimentos computacionais e análises comparativas entre os algoritmos para cada problema são apresentados. Também, uma análise do comportamento esperado para o problema da geração de Redes Candidatas em casos práticos foi realizada e será apresentada ao final deste capítulo.

6.1 Análise e Implementação de Algoritmos para o Problema da Árvore de Steiner

Já existem hoje na literatura diversos algoritmos que se propõem a resolver o problema da árvore de Steiner, sendo muitos desses aproximativos devido à complexidade NP-Difícil do problema. A Tabela 6.1 apresenta um comparativo entre os algoritmos de alguns dos trabalhos já publicados. Durante a pesquisa, foram implementados o algoritmo aproximativo de Takahashi e Matsuyama [41] e o algoritmo exato de Dourado, Oliveira e Protti [8]. Apesar do algoritmo de Vygen [42] possuir limite inferior próximo ao de Dourado, Oliveira e Protti [8], esse último foi escolhido por sua simplicidade.

Autores	Ano	Razão de aproximação	Limite inferior
Takahashi e Matsuyama	1980	2	$O(kn^2)$
Kou, Markowsky e Berman	1981	$2(1-1/l)$*	$O(kn^2)$
Wu, Widmayer e Wong	1986	$2(1-1/l)$*	$O(m\log n)$
Zelikovsky	1990	11/6	$O(mn+k^4)$
Vygen	2011	1	$O(nk2^{k+(\log_2 k)(log_2 n)})$
Dourado, Oliveira e Protti	2014	1	$O(n^2(n+m)+n^{k-2}+n\alpha)$

Tabela 6.1: Comparativo entre alguns algoritmos para resolução da árvore de Steiner. O parâmetro l representa o número de folhas na árvore ótima, α o número de árvores de Steiner enumeradas, n o número de vértices do grafo e k o número de terminais a serem interconectados.

6.1.1 Algoritmo de Takahashi e Matsuyama

O Algoritmo 2, proposto por Takahashi e Matsuyama [41], recebe como entrada um grafo qualquer e o conjunto dos vértices terminais, e fornece como saída uma árvore mínima de Steiner. O algoritmo é 2-aproximativo, portanto não garante fornecer a solução ótima. Seu limite superior é $O(kn^2)$, onde k representa o número de terminais e n o número de vértices do grafo.

Algoritmo 2: Takahashi e Matsuyama minimum Steiner-tree

Entrada: Grafo G, Conjunto de terminais W
Saída: Árvore T

1 $k \leftarrow$ tamanho de W
2 $T \leftarrow \emptyset$
3 remova um vértice qualquer de W e adicione em T
4 **para** *i de 2 até k* **faça**
5 encontre um vértice v de W tal que o custo do caminho de T até v em G seja mínimo
6 seja $P(T,v)$ o menor caminho entre T e v, adicione todos os vértices de P em T e todas as arestas de P em T
7 remova v de W
8 retorne T

De forma simples, o algoritmo em seu laço principal, tenta conectar o terminal da vez com qualquer outro terminal ou vértice já inserido na árvore utilizando qualquer algoritmo de resolução do problema do caminho mínimo, de forma que sempre o conecta com o que estiver mais próximo, seja este terminal ou não.

6.1.2 Algoritmo de Dourado, Oliveira e Protti

O Algoritmo 3 implementado foi proposto por Dourado, Oliveira e Protti [8]. É mais complexo do que o primeiro porque enumera todas as possíveis árvores de Steiner mínimas, dado um grafo e seu conjunto de terminais. O algoritmo é exato e portanto é exponencial dado o número de terminais da instância. Mas, o interessante é que tal algoritmo exato apesar de ser exponencial no caso geral, tem comportamento polinomial para um número fixo ou pequeno de terminais ou, mais formalmente, tal algoritmo possui atraso polinomial para gerar cada árvore de Steiner, uma vez que a quantidade dessas árvores pode ser exponencial mesmo para número de terminais igual a 3.

A primeira fase do algoritmo tem como objetivo gerar os templates ótimos, que serão explicados mais adiante. Na linha 2, o termo $d_G(u,v)$ representa a menor distância entre os vértices u e v no grafo G. Na linha 3, o conjunto de templates $\mathcal{T}^*$ é inicializado como vazio, bem como o custo dele é dado como ∞ na variável $d^S(W)$. Na linha 4, o conjunto R representa todos os subconjuntos de vértices não-terminais do grafo G que possuem $0, ..., |W|-2$ vértices. É feito um laço para iterar sobre todos os possíveis conjuntos R. Na linha 5, é construído um grafo G_{RW}, que consiste num subgrafo que contém somente os vértices de W e de R.

Na linha 6, são geradas todas as árvores geradoras mínimas de G_{RW}, denotadas pelo símbolo $\mathscr{T}$, que aqui são chamados de templates. Os autores recomendam o uso do algoritmo proposto por Eppstein [12]. No entanto não serão todas as árvores geradas que serão utilizadas, mas somente são usadas as cujos vértices de R possuem grau maior que 2. Nas linhas 7 a 11, caso o custo das árvores encontradas seja menor que o custo atual, apaga-se o conjunto dos templates e inserem-se as novas árvores (templates). Caso o custo das árvores encontradas seja igual ao atual, somente são inseridas as novas árvores. Assim sendo, o conjunto $\mathcal{T}^*$ ao final possui todas as árvores mínimas dos grafos G_{RW} criadas.

Agora na segunda fase do algoritmo, a partir dos templates ótimos, as árvores de steiner serão enumeradas. Nas linhas 13 e 14, uma estrutura $I_G[u,v]$ é criada para detectar todos os vértices intermediários z do grafo que podem compor um caminho entre os vértices u e v e que respeitam a distância mínima $d_G(u,v)$.

Nas linhas 15 a 19, esses vértices intermediários, que chamamos de z, agora serão identificados por camadas ℓ, que representam as distâncias, e armazenados na estrutura

Algoritmo 3: Dourado, Oliveira e Protti enumerate Steiner min-trees

Entrada: Grafo G, Conjunto de terminais W
Saída: Conjunto de árvores T

Fase 1: GERAÇÃO DOS TEMPLATES ÓTIMOS
calcule a distância $d_G(u,v)$ entre todos os pares de vértices $u,v \in V(G)$
inicialize $d^S(W) \leftarrow \infty$ e $\mathcal{T}^* \leftarrow \emptyset$
para *cada* $R \subseteq V(G) \setminus W$ *tal que* $|R| \leq |W| - 2$ **faça**
    construa G_{RW} usando as distâncias computadas na linha 2
    para *cada árvore geradora mínima* $\mathscr{T}$ *de* G_{RW} *onde* $grau_{\mathscr{T}}(v) > 2$ *para todo* $v \in R$ **faça**
        se $custo(\mathscr{T}) < d^S(W)$ **então**
            $d^S(W) \leftarrow custo(\mathscr{T})$
            $\mathcal{T}^* \leftarrow \{\mathscr{T}\}$
        senão se $custo(\mathscr{T}) = d^S(W)$ **então**
            $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{\mathscr{T}\}$
Fase 2: ENUMERAÇÃO DAS ÁRVORES DE STEINER
para *todos os pares* $u,v \in V(G)$ **faça**
    compute $I_G[u,v]$ como $I_G[u,v] = \{z \mid d_G(u,z) + d_G(z,v) = d_G(u,v)\}$
para *car par de vértices* $u,v \in V(G)$ **faça**
    defina a camada ℓ de $I_G[u,v]$ como o conjunto $\{z \in I_G[u,v] \mid d_G(u,v) = \ell\}$, para $0 \leq \ell \leq d_G(u,v)$
    para $0 \leq \ell < d_G(u,v)$ **faça**
        para *cada* $z \in I_G[u,v]$ *na camada* ℓ **faça**
            seja $N_{uv}(z)$ o subconjunto (lista) de vizinhos de z na camada $\ell+1$, considerando qualquer ordem na lista
para *cada* $\mathscr{T} \in \mathcal{T}^*$ **faça**
    seja $u_1v_1, u_2v_2, \ldots$ uma ordem das arestas em $E(\mathscr{T})$, execute ENUMERATE-ST(u_1v_1)

Procedimento: ENUMERATE-ST(u_jv_j)
para $0 \leq \ell < d_G(u_j,v_j)$ **faça**
    $d_\ell \leftarrow 0$
$P \leftarrow w_0 = u_j$
enquanto *P não está vazio* **faça**
    seja w_ℓ o vértice mais a direita de P
    se $\ell = d_G(u_j,v_j)$ **então**
        em $\mathscr{T}$, troque a aresta u_jv_j pelo caminho P
        se $j < |E(\mathscr{T})|$ **então**
            execute ENUMERATE-ST$(u_{j+1}v_{j+1})$
        senão
            imprima $\mathscr{T}$
        em $\mathscr{T}$, troque o caminho P pela aresta u_jv_j
        remova $w_{\ell-1}$ e $w_\ell = v_j$ de P
        $d_{\ell-1} \leftarrow 0$
    senão se $d_\ell < |N_{u_jv_j}(w_\ell)|$ **então**
        $d_\ell \leftarrow d_\ell + 1$
        $w_{\ell+1} \leftarrow d_\ell$-ésimo vértice de $N_{u_jv_j}(w_\ell)$
        em P, coloque $w_{\ell+1}$ à direita de w_ℓ
    senão
        remova w_ℓ de P
        $d_\ell \leftarrow 0$

$N_{uv}(z)$. Por exemplo, na Figura 6.1, temos nas extremidades os vértices A e D cuja distância entre eles é $d_G(A,B) = 4$. Os vértices intermediários z nesse caso, são os vértices B e C, estando B a uma distância 1 do vértice A ($\ell = 1$), e o vértice C estando a uma distância 3 do vértice A ($\ell = 3$). Note que para esse exemplo, a camada $\ell = 2$ não possui vértice.

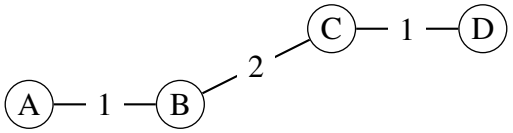


Figura 6.1: Exemplo de camadas do algoritmo entre dois vértices separados por $d_G = 4$.

Tendo agora todas essas estruturas definidas, o algoritmo executa em seguida procedimento ENUMERATE-ST para cada template $\mathscr{T}$ do conjunto de templates, recebendo como entrada uma primeira aresta. O procedimento basicamente substitui cada aresta do template pelos possíveis caminhos do grafo original e assim enumera todas as árvores de Steiner. A complexidade do algoritmo é $O(n^2(n+m) + n^{k-2} + n\alpha)$, onde m representa o número de arestas do grafo e α o número de árvores de Steiner enumeradas [8]. A Figura 6.2 mostra um exemplo de grafo, ao lado o único template gerado, e abaixo todas as árvores enumeradas a partir do template.

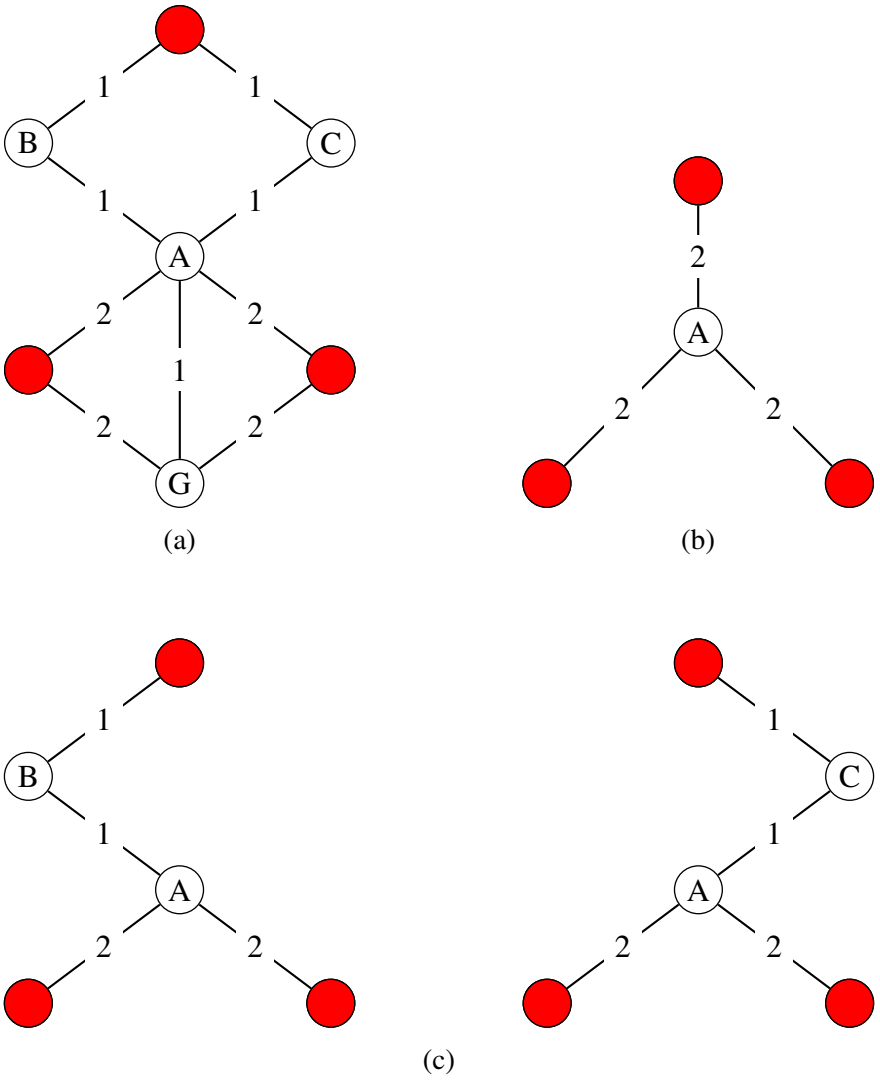


Figura 6.2: (a) o grafo original com os terminais marcados em vermelho. (b) o grafo G_{RW} de menor custo. (c) duas árvores de Steiner resultantes.

6.1.3 Heurística baseada no Algoritmo de Dourado, Oliveira e Protti

No algoritmo proposto pelos autores Dourado, Oliveira e Protti [8], são explorados todos os possíveis subconjuntos de roteadores que podem fazer parte da solução ótima. Para isso consideram os subconjuntos de $V(G) \setminus W$ com no máximo $k-2$ vértices que podem ser enumerados em tempo $O(n^{k-2})$.

A heurística proposta aqui consiste em analisar todos os vértices não-terminais e usá-los como roteadores, um por vez, e escolher aquele que gera o template $\mathcal{T}$ de menor custo. No entanto esse procedimento só é executado no máximo $k-2$ vezes, pois esta é a quantidade máxima de roteadores em uma árvore de Steiner mínima ótima. O Algoritmo 4 mostra o pseudo-código do algoritmo heurístico proposto.

Algoritmo 4: Heurística para árvore de Steiner

Entrada: Grafo G, conjunto W
Saída: Árvore de Steiner T

```
1  d_G ← todas as menores distâncias dos caminhos mínimos entre todos os vértices;
2  R ← ∅;
3  G_RW ← grafo completo com os vértices W ∪ R usando as distâncias de d_G;
4  T ← MST(G_RW);
5  melhor ← custo(T);
6  changed ← true;
7  enquanto changed = true e quantidade(R) < k − 2 faça
8      changed ← false;
9      para cada vértice v ∈ V(G) \ (W ∪ R) faça
10         G_RW ← grafo completo com os vértices W ∪ R ∪ v usando as distâncias de
             d_G;
11         T' ← MST(G_RW);
12         c ← custo(T');
13         se c < melhor e grau_T'(v) > 2 então
14             melhor ← c;
15             T ← T';
16             changed ← true;
17             v' ← v;
18     se changed = true então
19         R ← R ∪ v';
20 T ← T com caminhos mínimos expandidos;
21 retorna T;
```

Assim como no algoritmo original [8], a ideia geral consiste em utilizar uma matriz d_G com as distâncias dos caminhos mínimos entre todos os vértices do grafo G e construir templates $\mathcal{T}$, que são árvores geradoras mínimas de grafos completos G_{RW} formados com os terminais e os roteadores, utilizando as distâncias em d_G para os pesos das arestas.

Os vértices não-terminais são sempre visitados a cada iteração do laço **para** em uma ordem arbitrária, já que a ordem em que são visitados não altera o resultado. No último passo, o algorimo expande todos os caminhos mínimos contraídos do melhor template $\mathcal{T}$ encontrado para obter a árvore de Steiner da solução. A complexidade do Algoritmo 4 é dado pelo Teorema 4.

Teorema 4. *Seja G um grafo conexo e $W \subset V(G)$ tal que $|W| = k$ para um inteiro fixo positivo k. O Algoritmo 4 pode gerar uma árvore aproximada de Steiner em tempo $O(n^3 + nk^3)$, ou somente em $O(n^3)$.*

Demonstração. Primeiro computamos a matriz de distâncias d_G em $O(n^3)$ aplicando o algoritmo de Floyd-Warshall. Em seguida, no laço aninhado, temos o processo para determinar o conjunto de roteadores R, para com ele, construir a árvore de Steiner. Portanto, no laço **enquanto**, linha 7, o conjunto R é inicializado vazio e a cada iteração é adicionado um roteador de forma que ao final possua no máximo $k-2$ roteadores, portanto consome um tempo $O(k)$. A seguir, no laço **para**, linha 9, todos os vértices do grafo são visitados sem distinção (exceto os terminais e os vértices já inseridos em R), dado que ao final sabemos qual é o melhor vértice a ser adicionado ao conjunto R. Dentro deste laço, o passo de maior custo é a linha 11, dado que para cada grafo G_{RW} é extraída uma árvore geradora mínima deste. Como G_{RW} é sempre um grafo completo, seu número de arestas é sempre da ordem de k^2, e portanto, independente do algoritmo utilizado para extraír a árvore geradora mínima, a complexidade deste passo é de $O(k^2)$. Portanto, o laço aninhado é executado em tempo $O(nk^3)$, ou somente $O(n)$ dado que k é uma constante. Finalmente, o Algoritmo 4 possui complexidade computacional $O(n^3 + nk^3)$, ou, $O(n^3)$. □

Importante notar que caso a matriz de distâncias d_G seja fornecida na entrada, o algoritmo proposto passa a apresentar complexidade linear na ordem de $O(n)$.

A Tabela 6.2 apresenta os resultados dos testes computacionais realizados com as bases de teste de um *benchmark* conhecido [27]. Os algoritmos foram implementados em C++ e os experimentos executados em um servidor Ubuntu 18.04. Foram comparados os

valores das soluções encontradas pelo algoritmo (soma do peso das arestas da árvore de Steiner encontrada) com suas respectivas soluções esperadas pela base de testes. Para os experimentos foram utilizadas somente instâncias em que a solução ótima é conhecida.

	Esparsos com pesos aleatórios	**Completos com pesos aleatórios**	**Esparsos com pesos euclidia-nos**	**Completos com pesos euclidia-nos**	**Esparsos com pesos incidentes**	**TOTAL**
Instâncias testadas	60	27	14	14	395	514
Soluções ótimas encontradas	31	23	14	12	130	212
Porcentagem de ótimas encontradas	51,67%	85,19%	100%	85,71%	32,91%	41,25%
Média de aproximação	1,01	1,00	1,00	1,00	1,01	1,01
Pior aproximação encontrada	1,09	1,04	1,00	1,00	1,12	1,12

Tabela 6.2: Qualidade das soluções encontradas utilizando a heurística proposta.

O algoritmo proposto foi capaz de obter todas as soluções ótimas para grafos esparsos com pesos euclidianos. Para grafos completos, o algoritmo foi capaz de obter a solução ótima em mais de 85% dos casos. Excepcionalmente no caso de grafos esparsos com pesos aleatórios ou incidentes, a performance não foi tão boa: no primeiro caso, foi capaz de obter a solução ótima em um pouco mais da metade dos casos; e no segundo caso, somente em um pouco mais de 30% dos casos. A média empírica da razão de aproximação encontrada foi muito próxima de 1, o que aparenta ser um bom indicador para determinar a razão de aproximação.

Recordamos que esta heurística e os resultados acima foram publicados em um artigo nos anais do 3º Encontro de Teoria da Computação sediado em Natal-RN no ano de 2018 [33] e também foi apresentado um resumo no *VIII Latin American Workshop on Cliques in Graphs* [32].

6.2 Análise e Implementação de Algoritmos para a Geração de Todas as Árvores Geradoras Mínimas

Na linha 6 do Algoritmo 3, todas as árvores geradoras mínimas do grafo G_{RW} devem ser enumeradas. Conforme dito anteriormente, os autores recomendam o uso do algoritmo proposto por Eppstein [12]. Entretanto, tal algoritmo possui um alto grau de complexidade e foi objeto de amplo estudo. A partir desse algoritmo também foram estudados e implementados outros algoritmos. A seguir estão descritos os algoritmos estudados e os resultados obtidos.

Primeiramente, considere N como o número de árvores geradoras de um grafo G, e K como o número de árvores geradoras **mínimas** do mesmo grafo.

Os trabalhos de Gabow e Myers, Matsui, Kapoor e Ramesh, Shioura e Tamura apresentam algoritmos para enumerar todas as árvores geradoras de um grafo ([13],[34],[24],[39]). Tais algoritmos poderiam ser utilizados para resolver o problema da geração de todas as árvores geradoras mínimas de forma que as não-mínimas fossem descartadas ao final, restando assim somente as mínimas. Entretanto, tal estratégia demandaria um alto custo computacional desnecessário, considerando que o valor de N pode ser muito maior que K. O algoritmo proposto por Sörensen e Janssens é um pouco diferente dos demais porque fornece as árvores em ordem crescente de custo, algo que poderia ser uma solução um pouco mais elegante com menor custo, considerando que seria possível interromper o processamento assim que o custo da árvore em processamento fosse maior que o custo da anterior [40].

Alguns dos artigos com algoritmos para resolver o problema da geração de árvores geradoras mínimas ([12],[45],[43]) apresentam melhor complexidade computacional em comparação com os demais, como é possível ver na Tabela 6.3.

Algoritmo	**Ano**	**Resultado**	**Complexidade**
Gabow & Myers	1978	Árvores Geradoras	$O(n+m+N.n)$
Matsui	1997	Árvores Geradoras	$O(n+m+N.n)$
Kapoor & Ramesh	1995	Árvores Geradoras	$O(n+m+N)$
Shioura & Tamura	1995	Árvores Geradoras	$O(n+m+N)$
Sörensen & Janssens	2005	Árvores Geradoras (em ordem)	$O(N.m\log m+N^2)$
Wright	2000	Árvores Geradoras Mínimas	indefinido no artigo original
Yamada, Kataoka & Watanabe	2010	Árvores Geradoras Mínimas	$O(K.m\log n)$
Eppstein	1995	Árvores Geradoras Mínimas	$O(m+n\log n+K)$

Tabela 6.3: Algoritmos para gerar árvores geradoras e mínimas.

Neste trabalho, foram implementados quatro desses algoritmos apresentados, sendo que um deles está englobado no processo geral de resolução de outro.

6.2.1 Algoritmo de Yamada, Kataoka e Watanabe

São apresentados três algoritmos no artigo de Yamada, Kataoka e Watanabe [45], mas para o experimento, somente o segundo foi implementado. O algoritmo utiliza um *framework* de otimização combinatória e cortes no grafo para resolver o problema de enu-

meração de maneira recursiva. Basicamente o algoritmo considera dois conjuntos, F (*fixed*) como sendo o conjunto de arestas fixas que não podem ser retiradas da árvore atual e R (*restricted*) como o conjunto de arestas restritas, que não podem ser colocadas na árvore atual. Para o entendimento do Algoritmo 5, considere T uma árvore geradora mínima (F,R)-admissível, que é representada na forma $T = F \cup \{e^{k+1}, ..., e^{n-1}\}$, com $F = \{e^1, ..., e^k\}$.

Algoritmo 5: Yamada, Kataoka e Watanabe all MSTs

Entrada: Conjunto F, Conjunto R, Árvore T
Saída: Conjunto de árvores

```
para i = k+1, ..., n-1 faça
    encontre o conjunto Cut(e^i)
    encontre, se existir, um substituto e^{~i} ∈ S(e^i)
para i = k+1, ..., n-1 faça
    se e^{~i} existe então
        defina T_i ← T ∪ e^{~i} \ e^i e imprima T_i
        defina F_i ← F ∪ {e^{k+1}, ..., e^{i-1}} e R_i ← R ∪ {e^i}
        invoque All_MST_YKW(F_i, R_i, T_i) recursivamente
```

A seguir são explicados os conjuntos Cut(e) e $S(e)$. Considere T_0 como uma árvore geradora mínima arbitrária de G, e e uma aresta tal que $e \in T_0$, se retirarmos e de T_0, a árvore se divide em duas componentes conexas V_1 e V_2, assim o autor define Cut(e) como o conjunto de arestas que podem substituir e na árvore atual e reconectar as componentes V_1 e V_2, e também define o conjunto $S(e) := \{\tilde{e} \in \text{Cut}(e) \mid \tilde{e} \neq e, w(\tilde{e}) = w(e)\}$, ou seja, o conjunto de arestas que são candidatas a substituir e porque reconectam as componentes conexas e possuem o mesmo custo que e, não alterando assim o custo da árvore atual. É possível perceber que $S(e) \subseteq Cut(e)$. A complexidade do algoritmo é $O(Kmn)$.

Na Figura 6.3, é possível ver um grafo de exemplo e uma árvore geradora mínima correspondente ao lado. Ao fazer um corte na aresta (C,D), surgem dois componentes conexos. O conjunto Cut((C,D)) então possui duas arestas que reconectam os componentes: (B,D) e (C,E), no entanto, somente uma possui o mesmo custo de (C,D), que é (B,D), portanto, esta é a aresta a ser escolhida como substituta.

6.2.2 Algoritmo de Wright

Seja T_0 uma árvore geradora mínima arbitrária de G, e considere f como uma aresta de G que não esteja em T_0. Wright [43] prova em seu trabalho, que uma aresta $f = (u, v)$ é

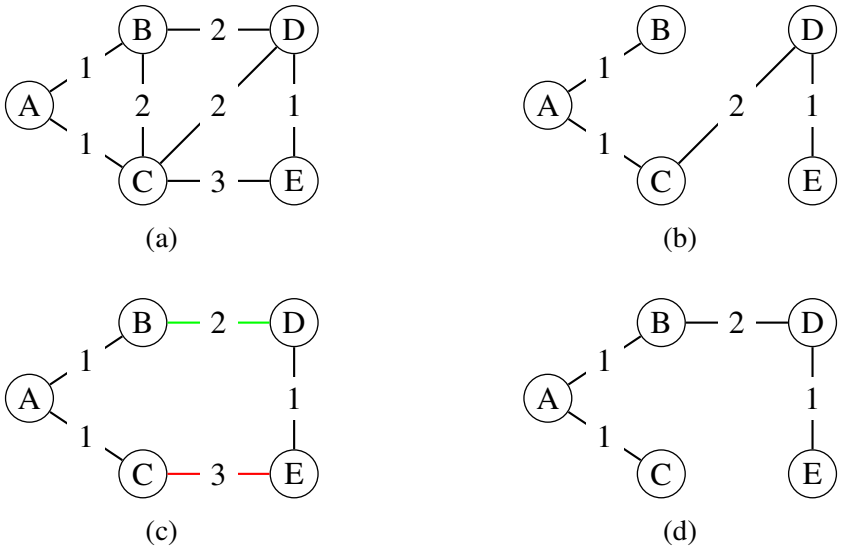


Figura 6.3: (a) Grafo original de exemplo. (b) Árvore geradora mínima arbitrária. (c) Arestas candidatas a serem substitutas de (C,D). (d) Nova árvore geradora mínima a partir da troca pela substituta (B,D).

considerada **elegível** para fazer parte de alguma árvore geradora mínima, se e somente se, existe uma aresta e no caminho de u a v em T_0 com custo igual ao de f, ou seja, somente se $w(e) = w(f)$. Esse teorema é utilizado, de certa forma, em todos os outros algoritmos de geração de árvores geradoras mínimas.

Para este algoritmo, o autor define como $[e]$ o conjunto de arestas elegíveis que podem substituir e em T_0 e gerar uma nova árvore geradora mínima; o termo $i([e])$ se refere ao número de arestas de e que precisam estar em qualquer árvore geradora mínima; e o termo $c([e])$ refere-se ao número de possíveis escolhas de $[e]$. Por exemplo, se considerarmos $[e]$ com 4 arestas e $i([e]) = 2$, então teremos 6 subconjuntos de tamanho 2 dado que $c([e]) = \binom{4}{2} = 6$ [43].

O Algoritmo 6 funciona da seguinte maneira: inicialmente recebe uma árvore geradora mínima T_0 de G como entrada e remove todas as arestas não elegíveis de G. Em seguida determina todas as classes de arestas elegíveis e os respectivos valores de $i([e])$ por inspeção em T_0. Finalmente, faz os conjuntos combinatórios, para cada classe, para posteriormente poder produzir todas as árvores geradoras mínimas de G. No entanto, nem toda combinação formam árvores válidas, então o autor faz uma pequena verifição que pode ser feita por dois métodos: basta testar se o grafo formado é acíclico e conexo, caso uma das verificações não seja válida, a resposta é descartada.

No artigo original, o autor não define a complexidade do algoritmo, no entanto aqui

Algoritmo 6: Wright all MSTs

Entrada: Grafo G, Árvore T_0
Saída: Conjunto de árvores

```
executa um DFS para pegar todas as arestas de G que não estão em T_0
para cada aresta f = (u,v) que não está em T_0 faça
    percorre o ciclo u a v em T_0
    para cada aresta do ciclo faça
        se w(e) == w(f) então
            adiciona aresta à classe [e]
une as classes com arestas de mesmo peso
para cada classe faça
    gera todos os subconjuntos combinatórios
seja L a lista de classes
c ← remove a primeira classe de L
GERA-ARVORES(c,L,T_0)

---

Procedimento: GERA-ARVORES(c,L,T)
T' ← T
remove as arestas originais de T' que serão substituidas pelas da classe c
para cada subconjunto da classe c faça
    insere arestas em T'
se T' for conexa e acíclica então
    se L = ∅ então
        imprime nova árvore T'
    senão
        c_2 ← remove primeira classe de L
        GERA-ARVORES(c_2,L,T')
insere c em L
```

fizemos uma simples análise. Consideremos S como o número de subconjuntos formados pelas classes e C como o número total de combinações arbitrárias das classes. Podemos recuperar todas as arestas não elegíveis em $O(m)$, obter as classes das arestas em $O(mn)$, formar os subconjuntos das classes em $O(S)$ e gerar e verificar cada árvore em (Cn). Podemos perceber, no entanto, que $C \gg S$ e $C \geq K$, então substituindo S por C temos $O(mn + Cn)$.

6.2.3 Algoritmo de Eppstein

O algoritmo proposto por Eppstein [12] apresenta o melhor desempenho computacional até então para resolver o problema da geração de árvores geradoras mínimas, entretanto, possui estrutura mais complexa. Seu principal mecanismo consiste em formar um grafo equivalente (EG) a partir de G realizando *sliding operations* de forma que todas as árvores geradoras de EG sejam equivalentes, uma a uma, com todas as árvores geradoras mínimas de G.

Uma *sliding operation* é definida da seguinte forma: considere duas arestas $e = (u, v)$ e $f = (v, w)$ que compartilham o mesmo vértice v, e suponha que $w(e) < w(f)$; para realizar tal operação, remove-se $f(v, w)$ do grafo e insere-se $f'(u, w)$ com o mesmo peso de f; mas tal operação somente é feita caso $w(e) < w(f)$. A Figura 6.4 exemplifica uma *sliding operation*.

Figura 6.4: Pequeno exemplo de aplicação de uma *sliding operation*.

Para formar o grafo EG, primeiramente é escolhido um vértice de forma aleatória para ser a raíz de T_0 e então são realizadas as *sliding operations* repetidamente pelas arestas e e f do grafo, contanto que $e \in T_0$ e que u esteja sempre mais próximo da raíz de T_0 do que v. Se implementado de forma direta, tal operação tem custo $O(mn)$, no entanto, o autor apresenta uma técnica para alcançar $O(m \log n)$. Para realizar isso, para cada aresta e em T_0, é definido o *heavy ancestor* de e, como a primeira aresta do caminho de e até a raíz que possui peso maior que e. Se for feita a *sliding operation* em cada aresta de T_0 diretamente com seu respectivo *heavy ancestor*, a complexidade é reduzida. O Algoritmo

7 mostra como tal procedimento é realizado de forma simplificada.

Algoritmo 7: Operação de sliding

Entrada: Grafo G, vértice u, vetor ℓ e *raiz*
Saída: Grafo Equivalente EG

```
1  marque u como visitado
2  para cada vértice v adjacente a u faça
3      se v não foi visitado então
4          if u = raiz then
5              A[0] = Aresta(u, v, custo(u, v));
6              SLIDING(G,v,0,raiz);
7              SLIDECHILDREN(u,v,0);
8          else
9              ℓ' ← BUSCABINARIA(A,custo(u, v),0,ℓ);
10             Aresta e = A[ℓ'];
11             A[ℓ'] = Edge(u, v, custo(u, v));
12             SLIDING(G,v,ℓ',raiz);
13             SLIDECHILDREN(u,v,ℓ');
14             A[ℓ'] = e;
```

O procedimento SLIDECHILDREN nas linhas 7 e 13, executa consecutivas *slinding operations* entre a aresta $u-v$ e todas as arestas $v-w$ adjacentes a v.

Antes de executar o Algoritmo 7 no entanto, todas as arestas que não farão parte de nenhuma árvore geradora mínima de G devem ser removidas de EG. Em seu trabalho, Eppstein não aborda este passo, mas fornece alguns trabalhos de referência, mas com a restrição de que tal passo necessita ser executado em tempo $O(m)$ para não comprometer a complexidade geral do algoritmo. Na implementação feita, utilizamos o trabalho de Bazlamacci e Hindi [3], que se enquadra no requisito de complexidade computacional exigido pelo autor.

O algoritmo proposto pelos autores, tem como objetivo, verificar se dada uma árvore geradora de um grafo, se está é mínima. O algoritmo faz uso de uma estrutura de dados chamada *Full Branching Tree* (B) introduzida por King [26]. Para formar tal árvore, é aplicado um algoritmo baseado no algoritmo geração de árvores geradoras mínimas de Boruvka, conforme mostra o Algoritmo 8 cuja complexidade é de $O(n)$.

A formação da *Full Branching Tree* é importante pois ela garante que, dados dois vértices (x,y), a aresta de maior custo entre x e y em T, tem o mesmo custo da aresta de maior custo entre x e y em B, e também que todos os nós folha de B correspondem, um a um, a todos os vértices de T, como pode ser verificado no exemplo da Figura 6.5.

Algoritmo 8: King Branching Tree

Entrada: Árvore T
Saída: Full Branching Tree B

```
1  V_B ← ∅
2  E_B ← ∅
3  para cada vértice v de T faça
4  |  criar novo vértice em V_B com f(v)
5  criar n árvores onde n é o número de vértices em T
6  enquanto n > 1 faça
7  |  para cada árvore t faça
8  |  |  encontrar menor aresta incidente a t em T
9  |  |  fundir t com árvore da aresta incidente escolhida
10 |  Seja A o conjunto de árvores que foram fundidas
11 |  para cada nova árvore formada A faça
12 |  |  adiciona novo vértice f(t) a V_B
13 |  |  para cada aresta a de A faça
14 |  |  |  adiciona nova aresta {f(t), f(a)} em E_B com custo igual à menor
   |  |  |    aresta incidente escolhida
```

Na segunda parte do algoritmo proposto por Bazlamacci e Hindi, é feita uma busca em profundidade na árvore B para determinar dentre todos os pares de vértices de T, qual o custo da maior aresta entre eles. O Algoritmo 9 apresenta o pseudo-código completo.

São utilizadas duas listas no algoritmo, BC e ML, a primeira serve para armazenar o custo da aresta anteriormente visitada, e a lista ML serve para guardar o índice da lista BC que indica o custo da maior aresta desde sua profundidade até a atual analisada no passo do algoritmo. Na linha 5, é feita uma busca binária na lista para determinar o índice do maior custo menor que c. A lista tem tamanho proporcional à altura da árvore B ($\log n_B$), e conforme dito anteriormente, o tamanho da árvore B é da magnitude da árvore T ($n_B \approx n$). Como é feita uma busca binária, tem-se que: $\log\log n_b \approx \log\log n \approx constante$.

Na linha 11, para determinar o *nca - nearest common ancestor* (ancestral comum mais próximo), foi utilizado o algoritmo proposto por Schieber e Viskin [38] que possui complexidade de $O(n+m)$. Assim, é possível na linha 12, descobrir qual a maior aresta entre u e o nca de (u,v) em B, e também a maior aresta entre v e o nca de (u,v) em B, assim, ao final tem-se qual a maior aresta entre u e v. O desempenho total final do algoritmo é de $O(m)$.

Após o grafo EG ter sido construído, o autor sugere o uso do algoritmo de Kapoor e Ramesh [24] para gerar todas as árvores geradoras de EG, que correspondem às árvores geradoras mínimas de G. No entanto, devido a alta complexidade de seus procedimentos,

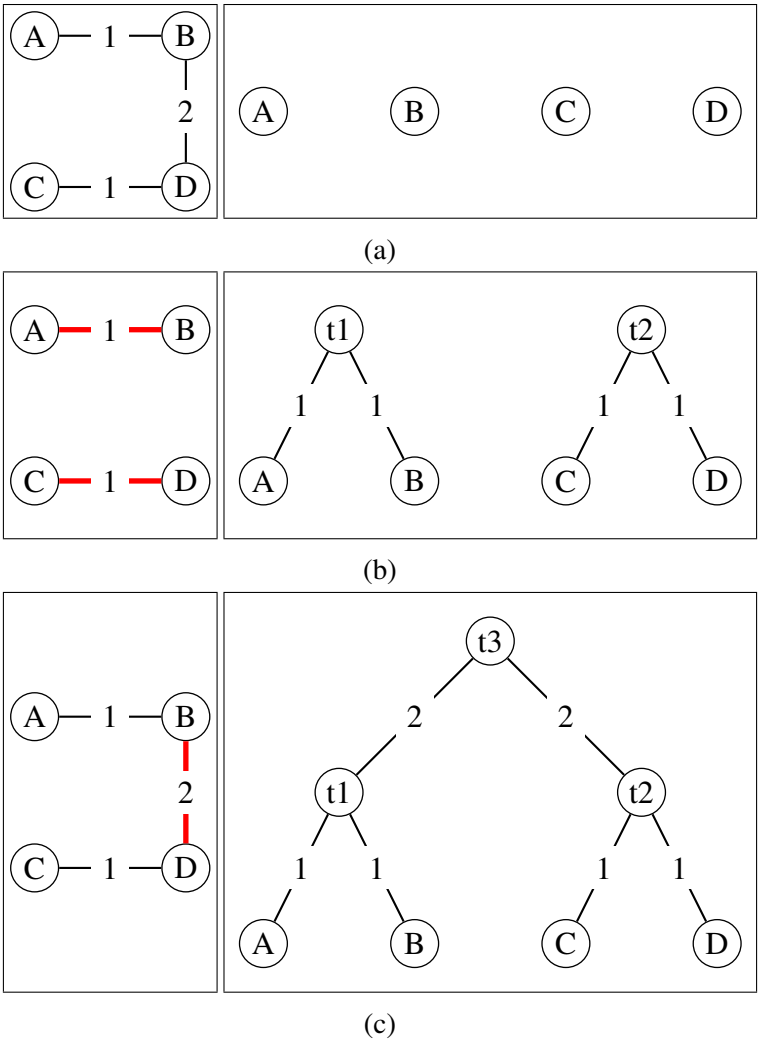


Figura 6.5: (a) Árvore T e ao lado somente os vértices para início da construção de B. (b) Seleção das menores arestas incidentes e formação dos novos vértices em B. (c) Seleção da menor aresta entre as árvores remanescentes e finalização de B.

foi tomada a decisão de implementar o algoritmo 8, proposto por Shioura e Tamura [39] que tem o mesmo desempenho mas com descrição mais simples.

O algoritmo considera que o vértice v_1 do grafo G seja a raíz, e que dada uma aresta e, os seus vértices das extremidades são escritos como $\partial^+ e$ e $\partial^- e$, assumindo que $\partial^+ e$ seja menor do que $\partial^- e$. O autor também necessita que algumas condições sejam válidas para que o resultado do algoritmo seja correto e obtido em bom tempo de execução. As condições são:

1. uma árvore DFS T_0 de G.
2. $T_0 = \{e_1, ..., e_{v-1}\}$, e qualquer aresta em T_0 é menor que seus descendentes.
3. cada vértice v é menor que seus descendentes em T_0.

Algoritmo 9: Verifica Aresta

Entrada: Full Branching Tree B, Árvore T, vértice u, Grafo G

```
1  marca u como visitado
2  d ← profundidade de u em B
3  se u não é a raiz de B então
4  |  c ← custo(u, pai(u))
5  |  idx ← BUSCA-BINARIA(c,0,d − 2)
6  |  BC[d − 1] ← c
7  |  para x de idx até d − 1 faça
8  |  |  ML[x] ← d − 1
9  |  se u é folha de B então
10 |  |  para cada vértice v adjacente a u em G faça
11 |  |  |  x ← BC[ML[profundidade(nca(u, v))]]
12 |  |  |  se x > custo(u, v) então
13 |  |  |  |  define x como maior aresta entre u, v em G
14 para cada vértice v adjacente a u em B faça
15 |  se v não foi visitado então
16 |  |  Verifica Aresta(B, T, v, G)
```

4. dadas duas arestas, $e,f \notin T_0$, $e < f$ somente se $\partial^+ e \leq \partial^+ f$.

Para garantir tais condições, o autor recomenda o uso do algoritmo de busca em profundidade (DFS) a fim de ordenar os vértices e as arestas da árvore gerada.

O Algoritmo 10 descreve os passos utilizados para resolver o problema. Na linha 4, para cada aresta, o conjunto $candi(e)$ representa uma lista de arestas que são candidatas a substituírem e na árvore. Tais arestas são candidatas se não estão na árvore inicial e se $\partial^- e = \partial^- f$ e $\partial^+ e \geq \partial^+ f$. Na linha 5, é montada a lista *leave* que nada mais é que uma lista das listas *candi* não vazias. Os demais procedimentos garantem a formação recursiva das árvores geradoras de forma que todas sejam impressas na tela somente uma vez. Este algoritmo possui complexidade de tempo de $O(m+n+N)$, onde N é o número de árvores geradoras do grafo.

Após a descrição detalhada de todos os passos, o Algoritmo 11 apresenta o pseudo-código completo do algoritmo de Eppstein da forma que foi implementado, com complexidade computacional $O(m+n\log n+K)$.

6.2.4 Experimentos Computacionais Comparativos

Os algoritmos foram implementados na linguagem C++ e os experimentos executados em uma máquina Intel i5, com 8GB RAM e sistema operacional Ubuntu 16.04 de 64 bits. A

Algoritmo 10: Todas as spanning trees

Entrada: Grafo G
Saída: Conjunto de árvores

obtem árvore T_0 por meio de DFS
ordena os vértices e arestas de T_0 para atender às condições
para *cada* $e \in T_0$ **faça**
    $candi(e) \leftarrow Can(e; T^0, V-1)$
$leave \leftarrow \{e \in T_0 | candi(e) \neq \emptyset\}$
imprime T_0
FIND-CHILDREN()
;

;
Procedimento: FIND-CHILDREN()
se $leave = \emptyset$ **então**
    return
$Q \leftarrow \emptyset$
$e_k \leftarrow$última aresta em $leave$
delete e_k de $leave$
enquanto $candi(e_k) \neq \emptyset$ **faça**
    $g \leftarrow$ última aresta de $candi(e_k)$
    deleta g de $candi(e_k)$, e adicione g ao ínicio de Q
    imprime a árvore $T^c \leftarrow T^p \setminus e_k \cup g$
    SUB-CHILDREN(e_k,g)
    desfaz a troca de arestas g e e_k
move tudo que está em Q para $candi(e_k)$
SUB-CHILDREN(e_k,e_k)
adiciona e_k no final de $leave$
;

;
Procedimento: SUB-CHILDREN(e_k,g)
se $candi(e_k) = \emptyset$ ***ou*** $\partial^+ g \leq \partial^+(\textit{primeira aresta em } candi(e_k))$ **então**
    FIND-CHILDREN()
    return
$f \leftarrow$ a aresta em T_0 com $\partial^- f = \partial^+ g$
se $candi(f) \neq \emptyset$ **então**
    $S \leftarrow \{e \in candi(e_k) | \partial^+ e < \partial^+ g\}$
    merge S em $candi(f)$
    FIND-CHILDREN()
    deleta todos os elementos de S em $candi(f)$
    $S \leftarrow \emptyset$
senão
    $candi(f) \leftarrow \{e \in candi(e_k) | \partial^+ e < \partial^+ g\}$
    insere f em $leave$
    FIND-CHILDREN()
    deleta f de $leave$
    $candi(f) \leftarrow \emptyset$

Algoritmo 11: Eppstein all MSTs

Entrada: Grafo G

Saída: Conjunto de árvores

1 cria árvore boruvka B de T_0 com algoritmo de King
2 calcula todos os **nca's** de B com algoritmo de Schieber e Viskin
3 obtém o custo da maior aresta entre todos os pares vértices de T em B com algorimo de Bazlamacci e Hindi
4 remove todas as arestas $\notin T_0$ que não serão parte de qualquer árvore geradora mínima
5 executa SLIDING(G,*raiz*,0,*raiz*)
6 enumera todas as árvores geradoras mínimas com algoritmo de Shioura e Tamura

performance dos três algoritmos descritos foi analisada em experimentos baseados nos experimentos propostos por Yamada, Kataoka e Watanabe [45]. Foram utilizados grafos completos e grafos aleatórios arbitrários, mas nesses últimos, o número de arestas foi fixado em $m = 3n$. Em relação à distribuição dos pesos das arestas, os experimentos foram divididos em duas categorias: grafos com arestas com mesmo peso e grafos com arestas com peso variando entre $[1, 10^L]$ (para $L = 2$ e $L = 3$). Os resultados dos experimentos são apresentados a seguir na Tabela 6.4 para grafos completos com arestas de mesmo peso, na Tabela 6.5 para grafos aleatórios onde o número de arestas é sempre $m = 3n$ e o peso das arestas é aleatório, e na Tabela 6.6 para grafos completos com peso das arestas aleatório.

Grafo	**AGMs**	**Wright**	**YKW**	**Eppstein**
K5	125	0,003	0,002	0,000
K6	1269	0,054	0,029	0,006
K7	16807	1,188	0,434	0,082
K8	262144	31,002	7,5482	1,335
K9	4782969	960,126	152,972	23,957
K10	100000000	*	3541,48	502,631

Tabela 6.4: Resultados para grafos completos com mesmo peso nas arestas.* tempo muito alto de execução.

Recordamos que a implementação destes algoritmos e os experimentos acima foram publicados em um artigo aceito pela revista Matemática Contemporânea volume 45 [31] e também foi apresentado um resumo no *VII Latin American Workshop on Cliques in Graphs* [30].

L	Vertices	AGMs	Wright	YKW	Eppstein
2	200	16	0,078	0,087	0,011
	400	48	0,350	1,720	0,026
	600	2048	6,467	76,091	0,042
	800	6144	23,276	230,59	0,06
3	200	1	0,065	0,051	0,011
	400	1	0,269	0,051	0,011
	600	2	0,614	0,755	0,042
	800	2	1,116	1,429	0,061

Tabela 6.5: Resultados para grafos aleatórios com $m = 3n$ e peso aleatório pras arestas.

6.3 Análise empírica para o STCEP como um R-KwS

Foram realizados experimentos nos bancos de dados de teste utilizados pelo trabalho de Coffman [4]: IMDb, Wikipedia e Mondial. Dois algoritmos para a resolução do problema de determinar a árvore de Steiner mínima foram estudados e implementados na linguagem Java em conjunto com um algoritmo para indexação das tuplas previamente feito por Oliveira [36]. O objetivo dos testes era gerar Redes Candidatas nos bancos de testes e avaliar a qualidade dos resultados. Os algoritmos foram descritos na Seção 6.1. A implementação do algoritmo de Takahashi e Matsuyama [41] serviu de suporte e foi utilizado em um trabalho de pesquisa do grupo de BDRI do PPGI/UFAM.

L	Grafo	AGMs	Wright	YKW	Eppstein
2	K20	4	0,004	0,001	0,001
	K40	120	0,046	0,056	0,006
	K60	176580	69,633	209,633	0,077
	K80	5971968	*	4218,73	0,167
3	K20	1	0,003	0,000	0,001
	K40	1	0,030	0,003	0,006
	K60	12	0,102	0,039	0,014
	K80	2	0,234	0,016	0,024
	K100	6	0,473	0,050	0,039
	K120	8	0,816	0,169	0,057
	K140	256	1,432	3,992	0,081
	K160	1152	2,681	8,282	0,104
	K180	208	2,902	1,940	0,133
	K200	1152	4,830	16,314	0,167

Tabela 6.6: Resultados para grafos completos com peso das arestas aleatórios. * tempo alto de execução.

Capítulo 7

Considerações Finais

Nesta dissertação foi apresentada uma caracterização teórica para o problema da geração de Redes Candidata que ocorre no processo de busca por palavras-chaves em bases de dados relacionais (R-KwS), onde foi possível demonstrar que tal problema pode ser modelado como uma variação do problema clássico de determinar uma árvore de Steiner em grafos com uma restrição adicional de coloração de arestas, com base em uma análise prévia de que o processo de geração de redes candidatas sucede um problema de enumeração de conjuntos das palavras-chave da busca bem conhecido (cobertura minimal) e abordado em trabalhos prévios. Tal problema foi denominado de problema de árvore de Steiner com arestas coloridas (do inglês, *Steiner Tree with Colored Edges Problem* - STCEP).

Também, foram apresentados aspectos de complexidade computacional para a aplicação prática, considerando a caracterização teórica apresentada, tanto para o caso geral quanto para aspectos práticos específicos da aplicação. Assim, foi apresentada a prova de NP-completude para o STCEP, bem como elaborada uma análise do pior caso. E, por fim, e apesar do desafio que esses problemas representam, o objetivo principal traçado inicialmente foi atingido, que consistiu em demonstrar que em cenários práticos o problema não possui complexidade computacional exponencial, onde é possível ser resolvido em tempo polinomial (dado que o número de palavras-chave da busca é fixo e geralmente muito pequeno).

Durante a pesquisa, para um melhor entendimento dos problemas envolvidos, foram estudados e implementados os principais algoritmos tanto para o problema clássico de árvore de Steiner em grafos, quanto para o problema gerar todas as árvores geradoras

mínimas (necessário para o problema anterior). Experimentos computacionais e análise comparativa dos algoritmos foram realizados, resultando em contribuições adicionais da pesquisa. Neste contexto, foi realizada a primeira implementação do algoritmo de melhor razão de aproximação para o problema da geração de todas as árvores geradoras mínimas, o algoritmo de Eppstein [12], seguida da primeira implementação do algoritmo exato para o problema clássico da árvore de Steiner, de atraso polinomial para um número fixo de terminais e que usa o algoritmo de Eppstein, proposto por Dourado, Oliveira e Protti [8]. Fruto disto, um trabalho foi aceito e apresentado no *VII Latin American Workshop on Cliques in Graphs* (LAWCG) [30] realizado em novembro de 2016 em La Plata, Argentina. Após o evento, um artigo sobre o trabalho apresentado foi publicado na revista Matemática Contemporânea da Sociedade Brasileira de Matemática [31]. Fruto da investigação dos algoritmos existentes, foi proposta uma nova heurística para o problema clássico de árvore de Steiner em grafos, simplificando o passo exponencial do algoritmo exato robusto de Dourado, Oliveira e Protti [8], que apresentou resultados competitivos com tempo polinomial de execução. Para este caso, foram aceitos e apresentados trabalhos para o III Encontro de Teoria da Computação (ETC - CSBC 2018) [33], realizado em julho de 2018 em Natal, e também, para o *VIII Latin American Workshop on Cliques in Graphs* [32], realizado em agosto de 2018 no Rio de Janeiro. Adicionalmente, dois artigos completos para periódicos estão em elaboração, um contemplando uma publicação científica com as contribuições na área de banco de dados e outra, para apresentar especificamente o problema proposto em grafos e aspectos de complexidade computacional.

Como possíveis extensões e trabalhos futuros, pretende-se realizar melhorias na heurística desenvolvida para o problema clássico de árvore de Steiner para, principalmente, provar a razão de aproximação do algoritmo e tentar, assim, validar teoricamente o que na prática demonstra ser muito promissor. Também, pretende-se continuar investigando o problema proposto, de árvore de Steiner com arestas coloridas (STCEP) de modo a verificar seu comportamento para classes específicas de grafos, e seu potencial de aplicação em outros contextos práticos.

Referências

[1] AGRAWAL, S., CHAUDHURI, S., AND DAS, G. Dbxplorer: a system for keyword-based search over relational databases. In *Proceedings 18th International Conference on Data Engineering* (2002).

[2] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Recuperação de Informação: conceitos e tecnologia das máquinas de busca*, 2nd ed. Bookman, 2013.

[3] BAZLAMACCI, C. F., AND HINDI, K. S. Verifying minimum spanning trees in linear time. In *Symposium on Operations Research* (Heidelberg, Germany, Sept. 1997), Springer-Verlag, pp. 139–144.

[4] COFFMAN, J., AND WEAVER, A. C. A framework for evaluating database keyword search strategies. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2010), CIKM '10, ACM, pp. 729–738.

[5] CORMEN, T. H., STEIN, C., RIVEST, R. L., AND LEISERSON, C. E. *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[6] CYGAN, M., FOMIN, F. V., KOWALIK, L., LOKSHTANOV, D., MARX, D., PILIPCZUK, M., PILIPCZUK, M., AND SAURABH, S. *Parameterized Algorithms*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[7] DIESTEL, R. *Graph Theory*, 5th ed. Springer, 2016.

[8] DOURADO, M., OLIVEIRA, R., AND PROTTI, F. Algorithmic aspects of steiner convexity and enumeration of steiner trees. *Annals of Operations Research 223*, 1 (2014).

[9] DOWNEY, R. G., AND FELLOWS, M. R. *Parameterized Complexity*. Springer-Verlag, 1999. 530 pp.

[10] DREYFUS, S. E., AND WAGNER, R. A. The steiner problem in graphs. *Networks 1*, 3 (1971), 195–207.

[11] ELMASRI, R., AND NAVATHE, S. B. *Sistemas de Bancos de Dados*, 4th ed. Pearson, 2006.

[12] EPPSTEIN, D. Representing all minimum spanning trees with applications to counting and generation. Tech. Rep. 95-50, Univ. of California, Irvine, Dept. of Information and Computer Science, Irvine, CA, 92697-3425, USA, 1995.

[13] GABOW, H. N., AND MYERS, E. W. Finding All Spanning Trees of Directed and Undirected Graphs. *SIAM Journal on Computing 7*, 3 (August 1978), 280–287.

[14] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[15] GILBERT, E. N., AND POLLAK, H. O. Steiner minimal trees. *SIAM J. Appl. Math. 16*, 1 (Jan. 1968), 1–29.

[16] HANAN, M. On steiner's problem with rectilinear distance. *SIAM J. Appl. Math. 14*, 2 (Mar. 1966), 255–265.

[17] HE, H., WANG, H., YANG, J., AND YU, P. S. Blinks: Ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2007), SIGMOD '07, ACM, pp. 305–316.

[18] HEARNE, T., AND WAGNER, C. Minimal covers of finite sets. *Discrete Mathematics 5*, 3 (1973), 247 – 251.

[19] HOLYER, I. The np-completeness of edge-coloring. 718–720.

[20] HRISTIDIS, V., GRAVANO, L., AND PAPAKONSTANTINOU, Y. Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29* (2003), VLDB '03, VLDB Endowment, pp. 850–861.

[21] HRISTIDIS, V., AND PAPAKONSTANTINOU, Y. Discover: Keyword search in relational databases. In *Proceedings of the 28th International Conference on Very Large Data Bases* (2002), VLDB '02, VLDB Endowment, pp. 670–681.

[22] HULGERI, A., AND NAKHE, C. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering* (Washington, DC, USA, 2002), ICDE '02, IEEE Computer Society, pp. 431–.

[23] KACHOLIA, V., PANDIT, S., CHAKRABARTI, S., SUDARSHAN, S., DESAI, R., AND KARAMBELKAR, H. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases* (2005), VLDB '05, VLDB Endowment, pp. 505–516.

[24] KAPOOR, S., AND RAMESH, H. Algorithms for enumerating all spanning trees ofundirected and weighted graphs. *SIAM J. Comput. 24*, 2 (Apr. 1995), 247–265.

[25] KARP, R. M. *Reducibility among combinatorial problems*. Springer US, Boston, MA, 1972, pp. 85–103.

[26] KING, V. A simpler minimum spanning tree verification algorithm. *Algorithmica 18*, 2 (1997), 263–270.

[27] KOCH, T., MARTIN, A., AND VOSS, S. SteinLib: An updated library on steiner tree problems in graphs. Tech. Rep. ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000.

[28] KOU, L., MARKOWSKY, G., AND BERMAN, L. A fast algorithm for steiner trees. *Acta Informatica 15*, 2 (Jun 1981), 141–145.

[29] LUO, Y., LIN, X., WANG, W., AND ZHOU, X. Spark: Top-k keyword query in relational databases. In *Proceedings of the 2007 ACM SIGMOD International*

Conference on Management of Data (New York, NY, USA, 2007), SIGMOD '07, ACM, pp. 115–126.

[30] MARTINEZ, J. G., DE FREITAS, R., AND DA SILVA, A. On the problem of finding all minimum spanning trees. In *VII Latin American Workshop on Cliques in Graphs* (November 2016).

[31] MARTINEZ, J. G., DE FREITAS, R., AND DA SILVA, A. On the problem of finding all minimum spanning trees. *Matemática Contemporânea 45* (2017), 97–105.

[32] MARTINEZ, J. G., DE FREITAS, R., DA SILVA, A., AND PROTTI, F. A strategy to select vertices as candidates for routers in a steiner tree. In *VIII Latin American Workshop on Cliques in Graphs* (August 2018).

[33] MARTINEZ, J. G., DE FREITAS, R., DA SILVA, A., AND PROTTI, F. Uma estratégia para selecionar vértices como candidatos a roteadores em uma árvore de steiner. In *Anais do III Encontro de Teoria da Computação* (Porto Alegre, RS, Brasil, 2018), SBC.

[34] MATSUI, T. A Flexible Algorithm for Generating All the Spanning Trees in Undirected Graphs. *Algorithmica 18*, 4 (Aug. 1997), 530–543.

[35] MELZAK, Z. A. On the problem of steiner. *Canadian Mathematical Bulletin 4*, 2 (1961), 143–148.

[36] OLIVEIRA, P. S. *Generation and Ranking of Candidate Networks of Relations for Keyword Search over Relational Databases*. PhD thesis, UFAM, 2017.

[37] SANTOS, V. F., AND SOUZA, U. S. Uma introdução à complexidade parametrizada. In *Anais da 34º Jornada de Atualização em Informática* (2015), SBC, pp. 232–273.

[38] SCHIEBER, B., AND VISHKIN, U. On finding lowest common ancestors: Simplification and parallelization. In *AWOC* (1988), vol. 319 of *Lecture Notes in Computer Science*, Springer, pp. 111–123.

[39] SHIOURA, A., AND TAMURA, A. Efficiently scanning all spanning trees of an undirected graph. *J. Operation Research Society Japan 38* (1995), 331–344.

[40] SORENSEN, K., AND JANSSENS, G. K. An algorithm to generate all spanning trees of a graph in order of increasing cost. *Pesquisa Operacional 25* (08 2005), 219 – 229.

[41] TAKAHASHI, H., AND MATSUYAMA, A. An approximate solution for the steiner problem in graphs. In *Math. Jap* (1980), pp. 24:573–577.

[42] VYGEN, J. Faster algorithm for optimum steiner trees. *Information Processing Letters 111*, 21 (2011), 1075 – 1079.

[43] WRIGHT, P. Counting and constructing minimal spanning trees. In *Bulletin of the Institute of Combinatorics and its Applications* (1997), pp. 65–76.

[44] WU, Y. F., WIDMAYER, P., AND WONG, C. K. A faster approximation algorithm for the steiner problem in graphs. *Acta Inf. 23*, 2 (Apr. 1986), 223–229.

[45] YAMADA, T., KATAOKA, S., AND WATANABE, K. Listing all the minimum spanning trees in an undirected graph. *Int. J. Comput. Math. 87*, 14 (2010), 3175–3185.

[46] ZELIKOVSKY, A. Z. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica 9*, 5 (May 1993), 463–470.